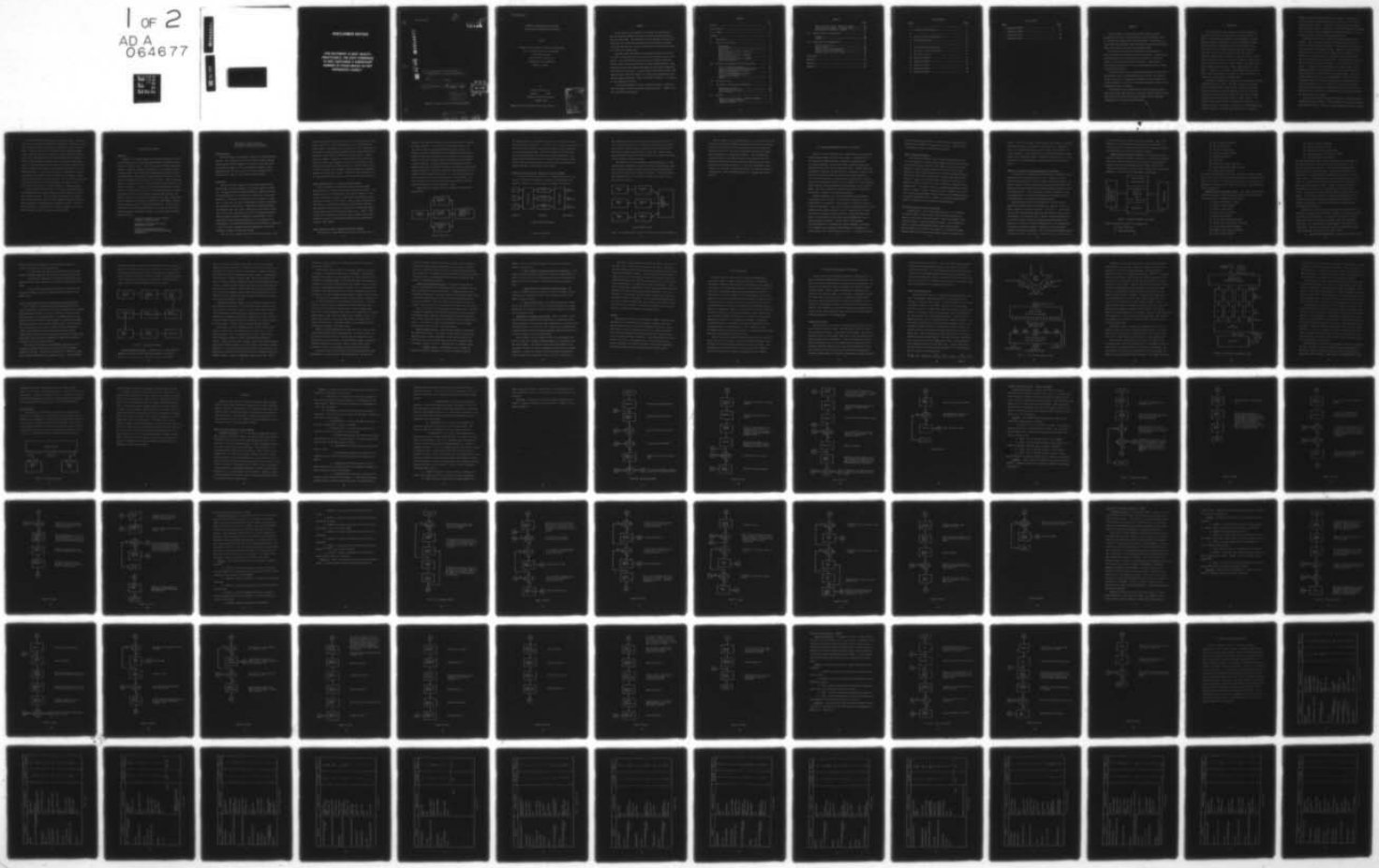
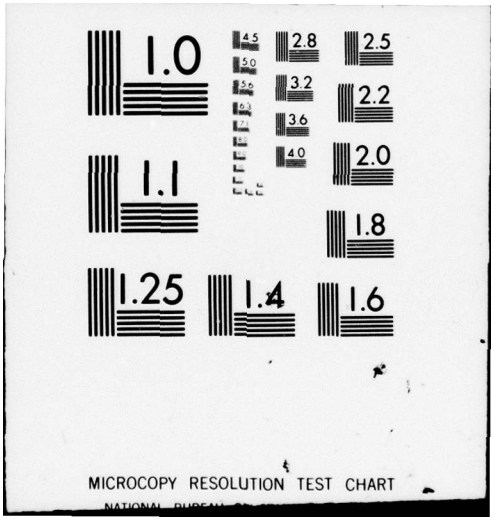
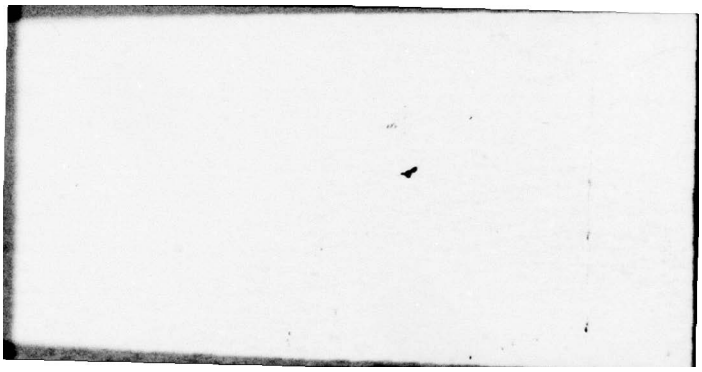
AD-A064 677 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB, OHIO F/G 17/4
A PARALLEL MICROPORCESSOR ARCHITECTURE FOR ELECTRONIC COUNTERMEASURES PROCESSING. DEC 78 HENRY, KENNETH L. MASTERS THESIS
UNCLASSIFIED REPT. NO. AFIT/GE/EE/78-26 NL

1 OF 2
AD A 064677




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART



DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AFIT/GE/EE/78-26

(1)

LEVEL II

AD A064677

DDC FILE COPY

(6) A PARALLEL MICROPROCESSOR ARCHITECTURE FOR ELECTRONIC COUNTERMEASURES PROCESSING.

(9) Master's THESIS,

(14) AFIT/GE/EE/78-26

(10) Kenneth L. Henry
Captain USAF

DDC
RECEIVED
FEB 15 1979
A

(11) Dec 78

(12) 159 pp.

Approved for public release; distribution unlimited

012 225

79 01 30 127

AFIT/GE/EE/78-26

A PARALLEL MICROPROCESSOR ARCHITECTURE FOR ELECTRONIC COUNTERMEASURES PROCESSING

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air Training Command

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

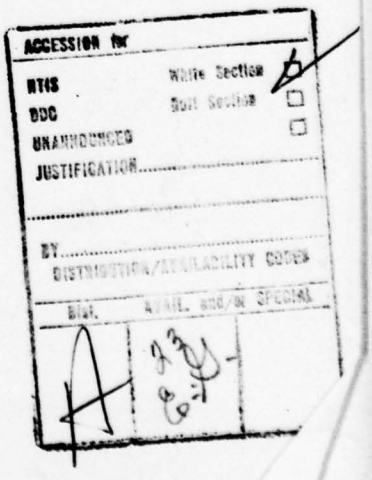
Kenneth L. Henry, B.S.

Captain USAF

Graduate Electrical Engineering

December 1978

Approved for public release; distribution unlimited.



Preface

In this thesis my main objective is to explore the application of parallel processing concepts to electronic countermeasures processing in Air Force ECM systems. The feasibility of these concepts has increased in recent years due to the rapid development of microprocessor technology. I sincerely hope that this report has accomplished its objective and serves as a stimulus for further study.

Many people have contributed to this report directly and indirectly. Mr. Daniel C. Holtz provided the initial idea and also served as my thesis sponsor. His professionalism and dedication are sincerely appreciated. Several members of the faculty at AFIT have been extremely helpful. Major Alan Ross has given much needed encouragement and many helpful suggestions in the wording of the report. A very thorough reading and criticism of the rough draft was rendered by Captain J. M. Borky whose comments were invaluable and deeply appreciated. Also, many thanks to Captain F. D. Kirschner for his reading and criticism.

A final acknowledgement of thanks is due my wife Karen. Without her help and support this thesis would not have been possible. I thank her for her patience and understanding.

Contents

		Page
Preface		ii
List of Figures		v
List of Tables		vi
Abstract		vii
I.	Introduction	1
II.	Parallel Architectures	4
	Background	4
	Multiprogramming	5
	Parallelism	5
	Single Instruction Stream - Single Data Stream Computers	6
	Single Instruction Stream - Multiple Data Stream Computers	6
	Multiple Instruction Stream - Multiple Data Stream Computers	8
III.	Electronic Countermeasures Processing Systems	11
	The EF-111A Jamming Subsystem	12
	The F-15 Tactical Electronic Warfare System	12
	AN/ALQ-131 Electronic Countermeasures Pod System Description	13
	The B-1 Radio Frequency/Electronic Countermeasures System	17
	Summary	23
IV.	The Soviet Threat	24
V.	The Parallel Microprocessor Architecture	25
	Architecture Overview	25
	Data Stream Steering Logic	26
	System Overloads	31
VI.	Software	33
	Normal Processing Software - Subroutine NORPROS	33
	Interrupt Processing Software - Subroutine INTSERV	41

Contents

		Page
	First Level Logic Array - Subroutine FSTRAM	47
	Second Level Logic Array - Subroutine SECRAM	56
	Data Extraction Subroutine - DATAOUT	67
VII.	Software Execution Time Analysis	71
	Analysis	89
	Summary	90
VIII.	Conclusions and Recommendations	91
	General Observations	91
	Overall Results	91
	Software Subroutine Improvements	92
	Potential Hardware Modifications	92
	Recommendations for Further Study	93
Bibliography		95
Appendix A		96
Appendix B		119
Appendix C		120

List of Figures

Figure		Page
1	.....	7
2	A Typical MIMD Architecture	8
3	.....	9
4	AN/ALQ-131 Computer Architecture	14
5	The RFS/ECMS Software	18
6	.....	26
7	First Level Data Steering Logic	27
8	One Sector's Data Steering Logic	29
9	The Master Processor	31
10	Subroutine NONPROS	37
11	Subroutine INTSERV	42
12	Subroutine FSTRAM	49
13	Subroutine SECRAM	58
14	Subroutine DATAOUT	68

List of Tables

Table		Page
1	Subroutine NORPROS	72
2	Subroutine INTSERV	75
3	Subroutine FSTRAM	78
4	Subroutine SECRAM	83

Abstract

The large number and wide variety of radio frequency emitters encountered in electronic warfare provide an enormous quantity of data for an electronic countermeasures system to process. Historically, all data has had to pass through a central processor for threat identification. Speed is the primary requirement for this ECM processor.

In this thesis the concept of parallelism was investigated as a method to increase the throughput of existing processors. Microprocessors were employed as the individual processing elements in a single data stream - multiple instruction stream architecture. Data steering or vectoring was accomplished via the use of programmed logic arrays stored in random access memories.

To test the feasibility of a parallel microprocessor architecture for ECM processing its basic building blocks were simulated. The necessary control logic, which is implemented in software in a master control processor, includes the capability to adapt the architecture to deal with the unpredictable nature of ECM data.

A fictitious scenario was developed to stimulate the data steering logic and test the master processor's capability to handle system overloads. Also, the actual ability of the system to process data was simulated through the use of fictitious radar parameters.

I Introduction

The large number and wide variety of radio frequency emitters encountered in electronic warfare provide an enormous quantity of data for an electronic countermeasures system to process. Emitter parameters must be processed at a rate of up to 300,000 to 400,000 pulses per second. Complicating this processing problem is the fact that the emitter identification parameters are clustered in high density groups in the frequency spectrum. An automatic electronic countermeasures system must be able to detect and identify threats, and apply jamming in a near real time manner. Historically, previous systems have all had a similar weakness. All data has had to pass through a central processor, either a human operator or a computer. In either case, the emitter recognition process has been the weakest and slowest element in the chain of events from receipt of a radar pulse to application of countermeasures.

The primary requirement for a system which is to process electronic countermeasures data is speed. The radar parameters which compose electronic countermeasures input data are: frequency, pulse width, pulse amplitude, pulse repetition frequency, and angle of arrival. Within the countermeasures processing system, this data is normally the output of a radio frequency (100 MHz - 50 GHz) receiver. For effective radar jamming it must be processed in real time, i.e., at the rate at which it is received. A problem arises when the processing system is placed in a real world application such as a tactical or

strategic aircraft which must penetrate enemy airspace. Typically, the rate at which data is being generated by the receiver exceeds the input capability of the processor. In the event that the processing system is unable to keep up with the incoming data, the data is either lost or delayed. This can result in serious consequences and the obvious solution is to use a faster processor or a more flexible processing system.

There are two basic ways to increase the speed and capability of a processor. One is to make the actual processor physically faster, i.e., use state-of-the-art digital technology. If this solution is available at an acceptable cost, the problem is solved. However, if the technology is unavailable or too expensive, another solution must be considered. An alternative to a faster processor is to improve the existing processor. This can be accomplished in two general ways. One is to make the processor work on more than one datum concurrently, i.e., parallelism in the data stream. If a processor can have several data in various phases of processing concurrently, it will be faster than a processor which must complete processing on one datum before it can accept another. The other general method to increase an existing processor's speed is to divide the processing algorithm into separate pieces, which are independent and can be executed concurrently, i.e., parallelism in the instruction stream. These concepts of parallelism can theoretically be used to make a basic processor up to N times faster if N parallel processing units are employed and ideal conditions are assumed.

Operational electronic countermeasures systems use various algorithms implemented in either software or digital hardware to process their received radar parameters [REF 9]. There are as many algorithms as there are systems.

These algorithms are well tested and are basically sound. Their weakness lies in the medium in which they are implemented. Significant improvement in their performance can be obtained if they can be made to operate faster. Aircraft such as the F-15 and the F-16 are examples of weapon systems which must operate in a tactical electronic warfare environment, that is, the number and density of the radars which they will encounter is relatively low, although the threat signals still occur in high density groups. This does not simplify the problem of providing adequate countermeasures, however, because they are small systems which cannot carry much processing hardware. Aircraft such as the B-1 and the B-52 are examples of strategic weapon systems which will encounter a large number, a great variety, and a high density of threat radars. This is partially compensated by the fact that they can carry more hardware than tactical systems. In either type of weapon system a faster processor is a better processor.

The main objective of this thesis is to investigate different types of parallel processing concepts and apply a suitable solution to the ECM data processing problem. The first step in this process is to analyze existing ECM systems and research various methods of obtaining parallelism. Next, a suitable architecture must be simulated and tested. Finally, the architecture will be stimulated with a ficticious scenario and ficticious radar parameters to test its data processing capability.

II Parallel Architectures

Background

Computation in a digital computer consists of the execution of a set of instructions which are normally grouped together and called a program. The program operates on data. There are many ways to define instructions and data. In general, however, an instruction is an operation, and data is operated upon. When discussing the performance parameters of a particular computational system, speed is usually measured in terms of how fast a block of instructions can operate on a block of data. When a system can operate on data input to it at a rate greater than or equal to the rate of arrival of the data, i.e., there is no backlog of data waiting to be processed, it is operating fast enough. In some cases instructions operating on one datum in a data stream cannot be executed before the next datum arrives. In this case the user has two basic choices, obtain faster hardware or software, or reorganize his system [REF 5: 1901-1909]. Whichever choice is made, the goal will probably be concurrency. The term concurrency refers to two or more events which occur in the same time interval, and with respect to computational systems, it can be achieved in varying degrees [REF 8:3].

> Concurrent execution of several different programs or multiprogramming.
>
> Input/output operations simultaneous with program execution.
>
> Multiple input/output operations to include data communications being executed simultaneously.

Concurrency of central processor operations in general either through parallelism or pipelining (confluence).

Multiprogramming

Multiprogramming is the concurrent execution of several different programs. This concept may be somewhat misleading, however, because even though more than one program is active at the same time, all of the programs must make use of the same hardware resources. Multiprogramming makes the best possible use of the existing hardware's capability but it doesn't really make the hardware faster.

Parallelism

A method to maximize computation speed without depending upon breakthroughs in device technology is to increase speed via parallelism. The ideal throughput (speed) potential of N identical computers (or microprocessors) is N times that of a single unit. Using microprocessors, a high degree of parallelism can be achieved at a modest cost. In some cases the entire microprocessor need not be replicated, and the concept of pipelining or a combination of logic replication and sharing of logic can be used. However, the most flexible architecture is one which is entirely parallel with N independent processors, which can support N different computations simultaneously. The N computations can be N different problems, or a partitioning of one large problem. One obvious difficulty which will definitely be encountered is that of timing or synchronizing these independent, free running processors to perform a particular function.

There are several methods for achieving parallel operation. They

depend upon replicating the instruction stream, the data stream, or both. A normal computer is a Single Instruction Stream - Single Data Stream (SISD) machine. Three types of parallel processors are: Single Instruction Stream - Multiple Data Stream (SIMD), Multiple Instruction Stream - Single Data Stream (MISD), and Multiple Instruction Stream - Multiple Data Stream (MIMD). In general, a parallel processor can exhibit parallelism in either the instruction stream, or the data stream, or both. The two most interesting and useful machines are the SIMD and the MIMD. The MIMD is best for doing independent tasks in parallel on independent computers and combining the results. The SIMD is most useful when used for computations which can be separated into functional blocks [REF 8].

Single Instruction Stream - Single Data Stream Computers

There is a method of increasing the speed of the SISD machine through confluence or concurrent operation on more than one datum. Each datum would be in a different phase of the machine's operation. There is still a bottleneck, however, in that only one instruction can be decoded per unit time in a SISD computer. Also, without replicating hardware, there is a limitation on the types of instructions which can be executed while various data are at various phases of operation, depending upon what hardware the instructions must utilize. There is only a narrow class of problems for which this type of computer can be used, and there are usually restrictive programming practices which must be enforced. [REF 5:1907]

Single Instruction Stream - Multiple Data Stream Computers

The major factor which determines the usefulness of this type of

computer is its ability to perform the desired computation in a sequence of vector operations. If single step in an operation depends upon the result of the last step, vector operations cannot be used. If, however, the identical operation is to be performed on multiple sets or streams of data, then this operation can be duplicated in multiple processors, and the data streams vectored to the processors, resulting in a SIMD architecture. An example of this type of operation is matrix multiplication, where an identical operation is performed on each row and column of the matrix. The proper row element and column element must be vectored to their respective processors, i.e., one processor might operate on all elements in each row and one column in a N X N matrix. In this case there would be N identical processors processing N simultaneous data streams, but only one instruction stream.

A SIMD computer is organized generally as shown in the following block diagram:

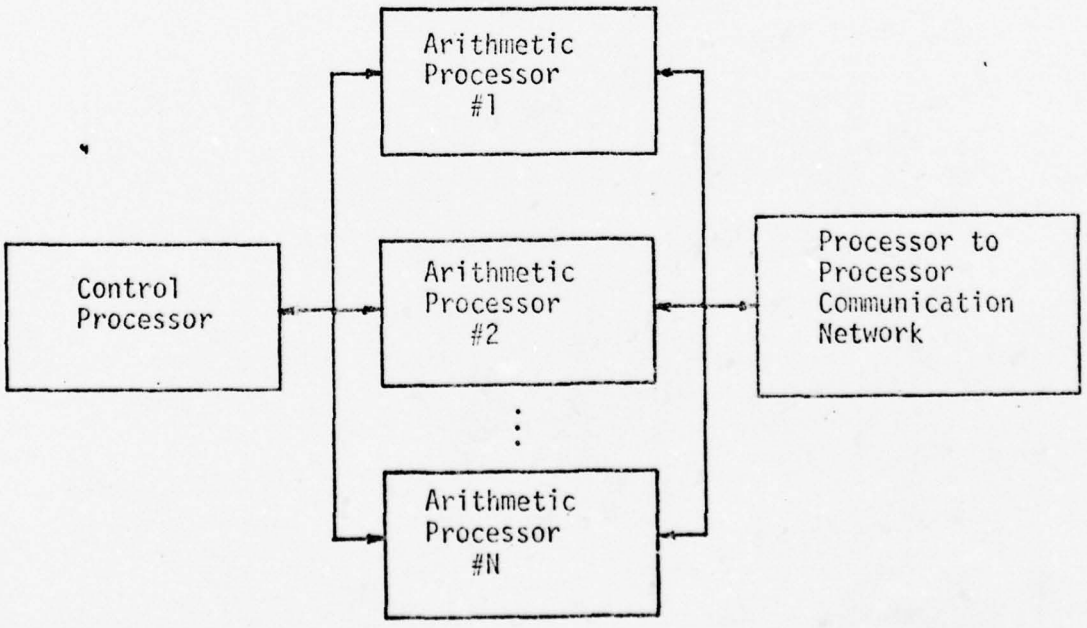
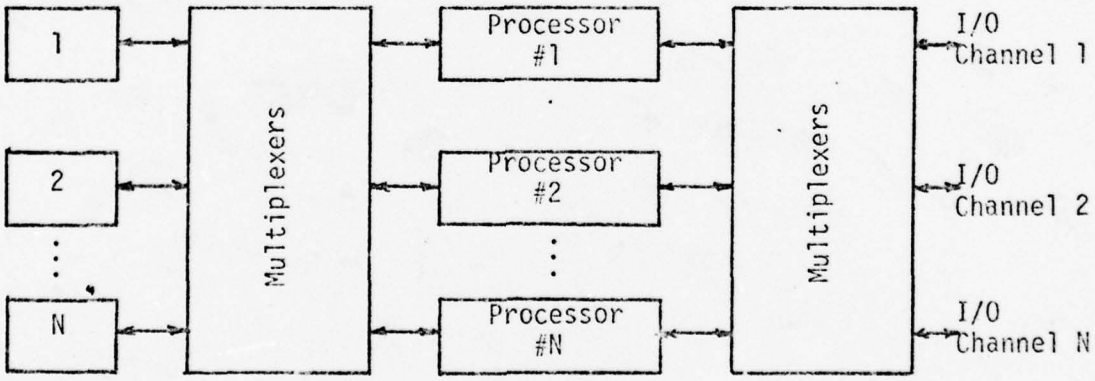


Figure 1 [REF 6:329]

The control processor is different than the others in that it can execute conditional branch instructions. It is the control processor which determines the order of the instruction stream and determines where each instruction is to be executed (vectoring). The control processor has the exclusive privilege of determining the instruction to execute next, which is the property which characterizes this as a single instruction stream machine. The fact that the instruction can be executed in various arithmetic processors makes this a multiple data stream machine.

Multiple Instruction Stream - Multiple Data Stream Computers

In the general MIMD computer there are N input/output channels (data streams) and N distinct processors (instruction streams) operating in parallel. Following is a block diagram of a typical MIMD computer:



A Typical MIMD Architecture

Figure 2 [REF 6:357]

Any input/output channel can be connected to any processor which in turn can be connected to any memory. The degree of flexibility exhibited by this machine is high, but the price which must be paid in the form of extra control hardware and/or software is also high. There are many potential conflicts which arise when resources such as memory are accessible to different processing elements.

One alternative to overly cumbersome control machinery is to interconnect several independent processors, each of which can execute independent instruction streams, but which share processors which perform all arithmetic and computational functional functions. This type of system was proposed by Flynn, et al. [REF 7: 251-286], and is shown in block diagram form in Figure 3.

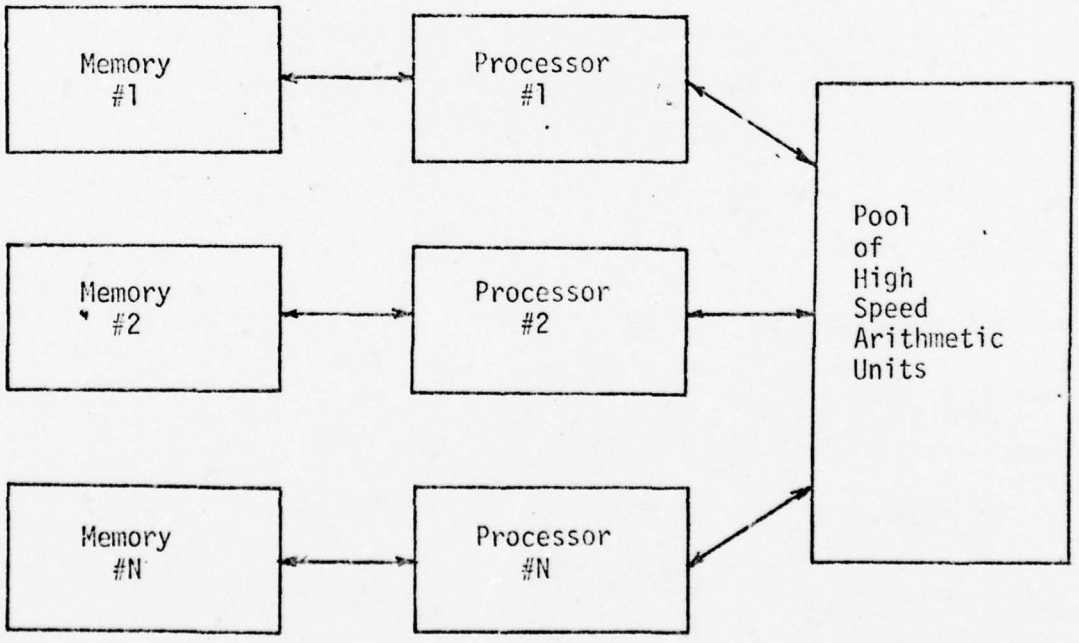


Figure 3 [REF 6; 361]

There is a tradeoff between decreased flexibility and increased simplicity.

The main problem areas in an MIMD computer are the partitioning of the overall operation to be performed into many processes that can be executed in parallel, the control of the separate processes, and the interaction between the processes. There are no established techniques for problem partitioning which can be universally applied. There are methods for resource sharing and scheduling which have been established for large computer systems and which can be applied to MIMD computers, but, again, the price is more control machinery. If one limits the degree of parallelism, i.e., the number of parallel partitions, the resource allocation, synchronization, and partitioning problems can be brought under control [REF 6].

III Electronic Countermeasures Processing Systems

There are many advanced electronic countermeasures systems currently being employed in modern Air Force weapon systems. Effective ECM activity requires rapid threat detection, reconnition, and application of a jamming technique. Most systems utilize a digital processor and associated software/firmware for automated operation. Digital processing also provides flexibility through software programmability, growth through modularity, and speed through its inherent computational capability. Air Force ECM systems vary in complexity from a self-protection internal electronic countermeasures system in the F-15 fighter to a single penetrator radio frequency surveillance/electronic countermeasures system in the B-1 bomber. The manner in which the essential processing is accomplished is not, however, as diverse as the system descriptions.

A general knowledge of Air Force ECM systems can be obtained by investigating several examples. The F-15 tactical electronic warfare system (TEWS) is an example of a fairly simple self-protection system for a specific application. A more general example is the AN/ALQ-131 electronic countermeasures pod which performs a similar function as the F-15 TEWS, but exhibits more capability and flexibility. The EF-111A jamming subsystem is a multi-mission, multi-function system designed for use in a tactical warfare environment. Finally, the B-1 radio frequency surveillance/electronic countermeasures system (RFS/ECMS) is an example of an advanced, power managed ECM system designed for protection of a single strategic bomber penetrating enemy airspace. A complete description

of each system would be redundant. Therefore, only the AN/ALQ-131 ECM pod and the B-1 RFS/ECMS will be described in detail. The EF-111A jamming subsystem and the F-15 TEWS will be summarized.

The EF-111A Jamming Subsystem

This system is designed for use in several tactical warfare roles. It can be employed as a standoff jamming system with the aircraft orbiting near the forward edge of the battle area (FEBA), it can be used for protection of a flight of attack aircraft with the EF-111A in an escort role, and it can be used for protection of a flight of attacking aircraft (and itself) with the EF-111A performing the attack along with the other aircraft. In general, the jamming subsystem performs the three basic functions of signal identification, threat recognition, and application of countermeasures. The manner in which it performs these functions is quite similar, in general, to the B-1 RFS/ECMS with this exception: the EF-111A uses a scanning superheterodyne receiver, whereas the B-1 RFS/ECMS employs a wide open channelized receiver. Control of a scanning superheterodyne receiver is discussed in the AN/ALQ-131 system description.

The F-15 Tactical Electronic Warfare System

Self-protection is this system's primary function. It is designed to recognize radio frequency signals, classify recognized signals as threats or non-threats, and provide either a radar warning signal to the aircraft operator, or apply ECM against the threat. The F-15 TEWS also employs a scanning superheterodyne receiver under computer control. Received signals are converted to IF, passed to an interface unit where they are digitized, and fed into the system's computer for signal/threat

analysis. Once in the computer, the individual pulse signals are extracted from an input storage buffer and built into pulse trains. These pulse trains are then compared against an a priori table of known threats for threat recognition. Next, a decision is made whether to provide a warning signal, initiate jamming, or both. The computer's final action is to tune the receiver to the next designated frequency band for further data collection.

AN/ALQ-131 Electronic Countermeasures Pod System Description

The AN/ALQ-131 is an advanced tactical electronic countermeasures pod designed for use on high performance aircraft. Modularity in system design provides flexibility to adapt to various mission requirements. The system is computer (software programmable) controlled which provides technique and mode control flexibility and reprogrammability to counter varied and changing threats. Also, this system is capable of operation in preset jamming modes, or in a power management mode with either an internal receiver/processor, or external radar warning receiver.

Functional organization of the AN/ALQ-131 is centered around the control and interface portion of the system which contains a programmable digital computer as the system controller. The receiver/processor module and the actual countermeasures modules for the different frequency bands are connected via a digital data bus. Operator interaction through cockpit controls includes selection of system operating mode (preset, semi-automatic, or automatic) and ECM technique selection. The control/interface module also contains hardware to generate up to 40 simultaneous waveforms for deception modulation. When a selected ECM technique requires a deception waveform, that waveform's time-variable values are transmitted

to the onboard terminal threat countermeasures modules. Also, in the automatic mode of operation the receiver/processor module sends time critical data (threat tracker pretrigger signals and blanking signals) to the countermeasures modules via a high speed data bus.

AN/ALQ-131 Digital Computer Description. The computer is a third generation design with a general register file, a microprogram controller, and a short, powerful, multi-format instruction set. The instruction repertoire features direct addressing of all memory locations and instruction modification for register data, immediate data, and indexing. Following is a block diagram of the computer's architecture:

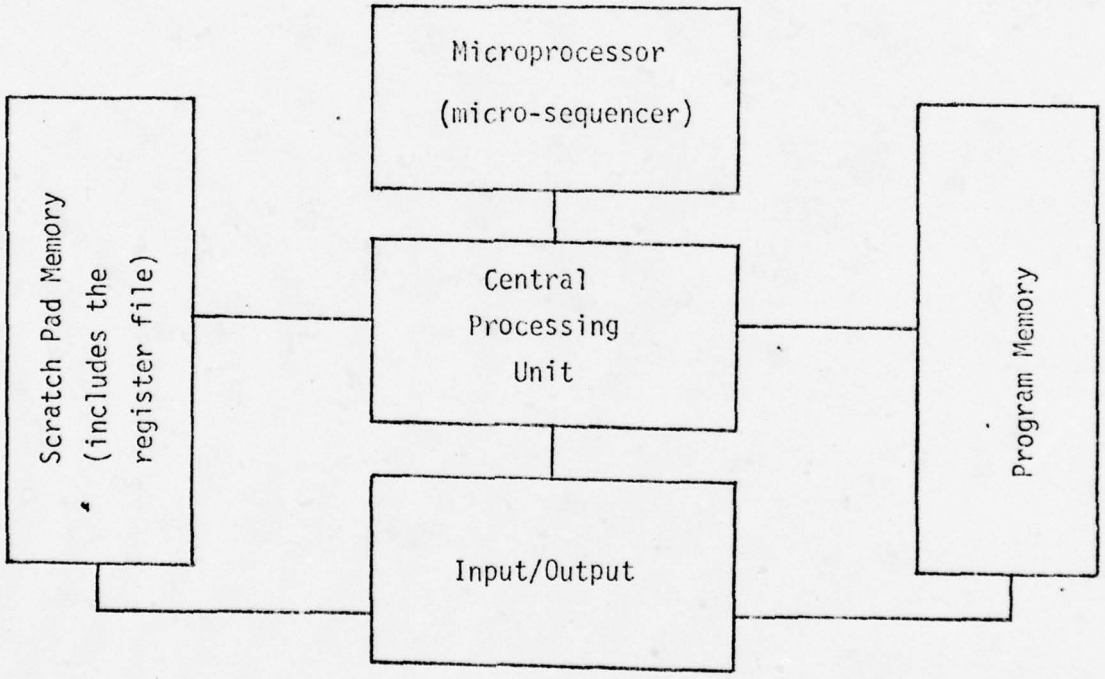


Figure 4 - AN/ALQ-131 Computer Architecture

Some of the general features of the computer are:

1) parallel operation

2) single address logic

3) two's complement arithmetic

4) 16 -bit word length

5) 16 & 32 bit instructions

6) 16 general purpose registers

7) microprogram control

8) 2 MHz clock

9) 70 nanosecond memory access time

10) direct memory access (DMA) capability

11) seven priority interrupts

As a part of the AN/ALQ-131 system, the computer has 3K words of random access memory (RAM), and 13K words of programmable read only memory (PROM). The main I/O consists of a 16 bit parallel port which is also used as a DMA channel in conjunction with several control lines. The computer also employs interrupt I/O.

AN/ALQ-131 Software Functional Summary. The computer programs which reside in the system s receiver/processor module control its operation. The functions which are controlled are as follows:

1) system initialization (EXEC2)

2) executive control (EXEC3)

3) receiver control (RCM)

4) pulse train sorting (PTS)

5) signal analysis (TWS)

6) launch status determination (LSD)

7) maintenance of active threat file (ATT)

8) receiver VCO calibration (CALIB)

9) analysis tracker set-on (TRKSETON)

10) tracker update (TRKUPDAT)

11) tracker assignment (TRKASSIN)

12) process pod commands (INTFAC)

13) reacquire tracking of signals (ROD)

14) tracking/guidance beam correlation (CORR)

15) self test monitoring (SELFTEST)

The receiver/processor acquires signals from threat radars, analyzes them for identification, and sends the required data to the jammers. The receiver, under software control, is tuned through preprogrammed threat frequency ranges to search for signals. When activity is detected, the receiver dwells for a sufficient time interval for the processor to do its signal analysis. Signal parameters are fed into the processor via the DMA channel for a predetermined amount of time dependent upon expected emitters' pulse repetition frequencies (PRFs). Analysis includes determination of pulse repetition interval (PRI) through examination of pulse time of arrival. Scan rate is measured by the TWS subroutine to help resolve ambiguities. Initial identification is attempted using frequency and PRI by comparing measured values against known expected threat parameters. The software also maintains an active threat table which contains data on each active threat detected.

In general, the software initiates the generation of waveforms by the electronic countermeasures (ECM) hardware. Software also sends beam steering information to the ECM modules. These modules contain RAMs in a control memory matrix which store waveform generation information, and control information, i.e., channel number, center frequency, and bandwidth (for noise modes). Application of countermeasures by the system software uses a power management scheme which is detailed as follows:

1) Technique Selection/Activation - Acquisition and identification

of threats, selection of techniques, and allocation of resources in accordance with the actual threat environment.

2) Time Gating - PRF tracking of selected emitters and interleaving of deception modulation and techniques on a pulse-to-pulse basis.

3) RF Set-On - Measurement of the threat signal frequency and adjustment of the jammer frequency to concentrate jammer activity on the threat.

4) Scan Set-On - Measurement of the threat's scan period and adjustment of deception modulation (wobbulation) to a harmonic of the measured scan.

The B-1 Radio Frequency/Electronic Countermeasures System (RFS/ECMS)

The B-1 RFS/ECMS is an advanced digitally controlled power and resource managed system, employing several levels of computer control ranging from controls and displays for operator interaction to real-time microprocessor control of the jamming hardware. To achieve flexibility and responsiveness, the RFS/ECMS has been designed as a digitally controlled, software programmable, power managed, modular system. Its main features are a channelized receiver which is wide open to the RF environment, a central digital processor which performs such functions as threat identification and selection of countermeasures, and digital jamming logic which handles the power and time management of applying countermeasure techniques to the transmitted waveforms.

The software which resides in the RFS/ECMS preprocessor has been designed to provide the system with as much flexibility and growth potential as possible. The key design features are: modularity, higher order language (JOVIAL/J3B), and a tabular data base. Pulse descriptor

words from an encoder are placed in a pulse word buffer in the computer's memory via direct memory access and the preprocessor assumes responsibility for the ECM processing from that point, continuing until a jamming technique and associated parameters have been assigned. The various functions of the preprocessor flight software are described in the following paragraphs, and these functional elements are shown in Figure 5.

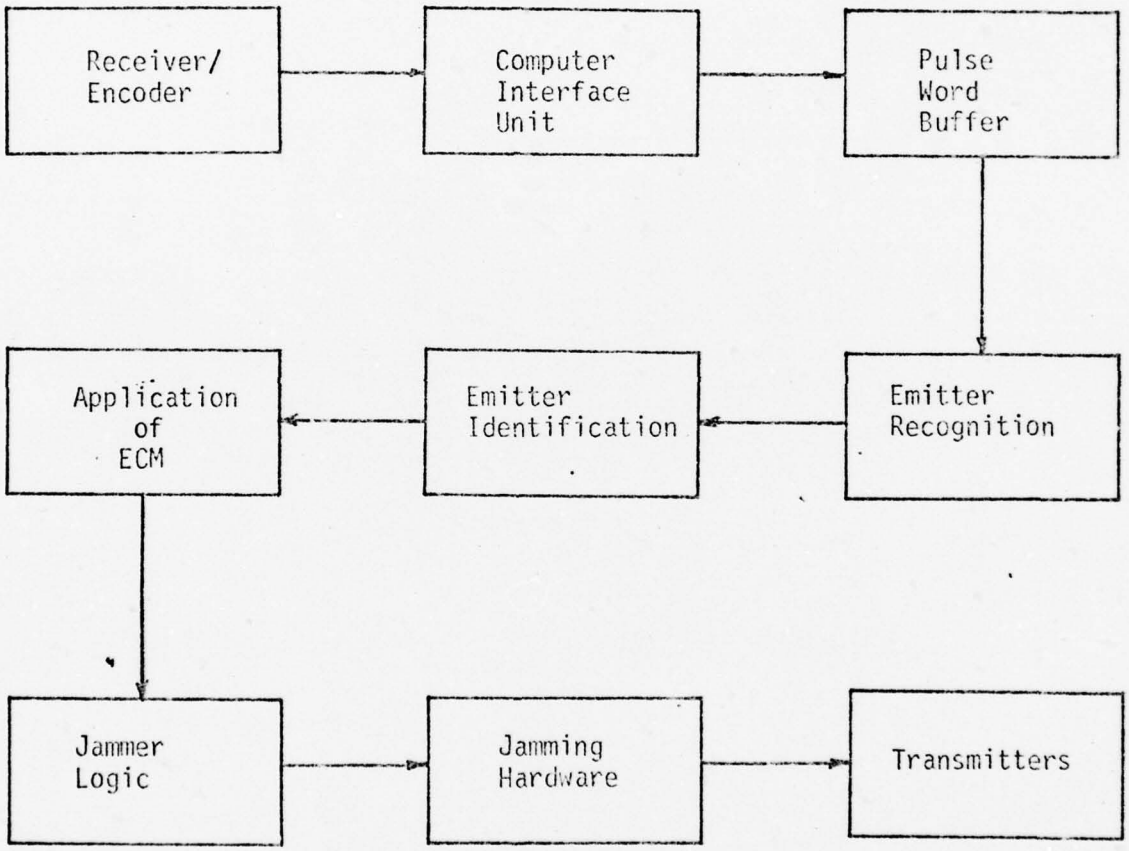


Figure 5. The RFS/ECMS Software

Emitter Recognition Function. As pulse descriptor words enter the computer's pulse word buffer from the receiver/encoder it processes the emitters' parameters. The digitally encoded pulse descriptor

words are composed of frequency, pulse width, angle of arrival, pulse amplitude, and time of arrival. The process of pulse train de-interleaving and parameter calculation involves the formation of a working file where pulse descriptor words are grouped by parameter matching. Parameters used for matching are frequency and pulse width. The software then uses the time of arrival of successive matching pulses to calculate pulse repetition interval. Eventually, pulse amplitude and time of arrival will be used to determine scan rate, however, identification can be made in the majority of cases based upon frequency, pulse width, and pulse repetition interval. Capability is also included for handling multibeam emitters and frequency/time of arrival ambiguity resolution.

Emitter Identification Function. Once an emitter's parameters have been calculated, this function is responsible for identifying that emitter. The preprocessor's data base contains an a priori table known as the Alarm File which contains the parameters of all known threats as well as the countermeasures technique to be applied. Another part of this function is a priority subroutine which affixes a priority number to a threat according to its leathality. Priority is used to resolve resource conflicts.

Pulse width is the first parameter used for emitter identification. Since it is not measured very accurately, it is essentially a coarse sorting parameter. Next, a frequency match is sought. Those emitters whose frequencies overlap are further sorted using pulse repetition interval. These three parameters; frequency, pulse width, and PRI, uniquely identify most emitters; however, scan rate is used to resolve any remaining ambiguities. If a parameter overlap can still exist after using scan rate, the potentially ambiguous alarm file entries are tied together using a scan ambiguity pointer. Because the ambiguous emitters are so alike, identi-

fication is made by arbitrarily identifying the emitter as the one with the greatest lethality.

In the dynamic stripping process the computer identifies an emitter and assigns an associative tracker if one is available. As part of the assignment the computer tells the tracker when to admit more pulses from an emitter for parameter update. When system time equals the time when the computer wants more pulses from an emitter, the trackers inspect the pulse descriptor words entering the computer interface unit. If one matches the frequency or the angle of arrival of the emitter on which the information was desired, the word is tagged with the tracker's number and allowed to pass.

The second way to block pulse traffic from the computer is through the use of static stripping logic in the computer interface unit. This logic works in a similar fashion as the dynamic stripping logic, but does not have the capability to track time of arrival. Static stripping logic consists of a bank of parameter limits which can be loaded into the "compare between limits" logic. Pulse descriptors are not allowed to pass if they fall between frequency limits, pulse width limits, amplitude limits, or angle of arrival limits. The various limits can be tightened or relaxed according to the degree of blocking desired.

Dynamic stripping through the use of the associative trackers is the optimum processing method for the preprocessor. Associative trackers are also used for time of arrival or frequency ambiguity elimination, main beam detection, receiver control, and time synchronization of the jamming waveforms with the received pulses. Static stripping logic is used primarily for pulse traffic reduction during an overload condition.

The emitter identification function also has the capability of mode determination for emitters with a multimode capability. The mode file

section of the alarm file is used for this function. Mode file entries also contain jamming technique information where a change is required. Other files subsidiary to the alarm file are the PRI file which contains the expected PRI's for emitters with more than one, the beam file for multi-frequency emitters, and the downlink file which contains jamming information for missile downlink signals.

Pulse Traffic Regulation Function. The RFS/ECMS must be able to handle a high density emitter environment; however, the computer does not have sufficient pulse handling capability. To reduce the pulse traffic through the preprocessor, the computer interface unit contains digital logic to selectively block pulse descriptor words from emitters which have already been identified. Pulse descriptor words are allowed to enter the computer only on new threats or to update the parameters of an old threat.

There are two methods of blocking pulses from the computer. The first is dynamic stripping which uses digital logic called associative trackers. Associative trackers consist of frequency memories, direction memories, and a content addressable memory which provides the logic to track time of arrival. The trackers are used in conjunction with a bank of compare between limits logic which does the actual blocking.

Receiver Control Function. Control of the receiver is exercised by the preprocessor software. The software obtains control of the receiver through the use of variable attenuators in the receiver itself or the associative trackers which contain logic to turn off a receiver channel or sub-band. Receiver control is established for the following reasons:

a. Operator Control. The operator may want to change the receiver's sensitivity threshold. The software accomplishes this by

sending a new set of threshold commands to the receiver through the computer interface unit.

b. Jammer Leakage and Lookthrough/Isolation Performance. If the isolation between the receive and transmit antennas is not sufficient to permit the receiver to have an unobstructed view of the environment, the software can raise the receiver's sensitivity threshold on a dynamic basis.

c. Special Search for CW and Pulse Doppler Emitters. The software, through the associative trackers, can look at selected receiver channels at an intermediate sensitivity for the purpose of CW or Pulse Doppler emitter detection.

d. Deferred Threat Search. The software can sequence through the receiver channels one at a time (all others switched off), at a high sensitivity, for the purpose of early detection of emitters or the detection of low power signals.

Application of Countermeasures Function. After the emitter identification function has matched a detected emitter's parameters with the a priori parameters in the alarm file, this function makes an entry in a file of detected emitters. This emitter file is then used by the application of countermeasures function for emitter parameter updates and transmission of emitter parameters to the jamming hardware.

Direct interface with the jamming hardware is via the jammer file. This file contains the jamming technique parameters to be transmitted to the actual hardware. The parameters which make up the jammer file are contained in the alarm file and are adapted using actual measured emitter parameters. In addition, a jammer file entry contains the associative tracker address for the tracker assigned to the emitter being jammed.

The specific jamming hardware with which the software interfaces is the jammer logic. Information is conveyed in two formats. One format, the jammer allocation logic load format, tells the logic how to set up the RF sources for jamming an emitter. The other format, the waveform generator format, tells the waveform generator what modulation to put on the transmitted jamming signal. When the required parameters are transferred to the jamming logic, it is necessary to "set-on" the hardware. Set-on refers to close loop frequency set-on of the voltage controlled oscillators and close loop set-on of the voltage variable attenuators, both in the transmitters. The software sends out commands to tune these devices, the transmitters send out a test pulse, the RFS receives and encodes the output, and the software compares the actual output with the required output and sends out a new command. This process is repeated until the desired accuracy is achieved for frequency and amplitude set-on.

Summary

Comparison of these four system descriptions yields a common processing algorithm for initial ECM data processing. Although each system employs different hardware, the processing schemes are similar. Data is collected by a receiver, digitized, and passed to a central processor. The computer then attempts to de-interleave this data stream into pulse trains associated with separate emitters. Threat identification is accomplished, in all cases, via matching with an a priori table of threat emitter parameters, and, finally, a decision is made on the desirability of electronic countermeasures application against detected threats.

IV The Soviet Threat

The sheer quantity of data which an electronic countermeasures system must handle is enough to swamp most processors. Complicating the problem is the fact that individual pulse descriptor words consisting of frequency, pulse width, pulse amplitude, angle of arrival, and time of arrival must be sorted according to the radar they were received from. This process, referred to as pulse train de-interleaving, is one of the crucial steps in electronic countermeasures processing. Various de-interleaving algorithms are used, but most involve grouping individual pulses by common frequency and pulse width (PW) until pulse repetition frequency (PRF) can be calculated. These three parameters are then used to identify an emitter and select a countermeasures technique. When a hostile radar can be identified uniquely via frequency, PW, and PRF the application of countermeasures process is a straightforward sequence. If, however, parameters from two or more radars are similar or overlapping, the identification process can be quite complex. In the latter case, pulse descriptor words must be processed as rapidly as possible, making a parallel processing system ideally suited for the task.

Appendix B contains the details on radar parameters which a USAF ECM system can expect to encounter. It serves to introduce the Soviet threat and show the inter-relationships between their radars' parameters. Due to Appendix A's security classification it is not normally included as part of this thesis, but if it were it could be inserted as Chapter 4.

V The Parallel Microprocessor Architecture

The main purpose of the parallel microprocessor architecture is to process electronic countermeasures data in a pseudo real-time mannar. In this case, pseudo real-time means fast enough that the microprocessor network can process the digital data being output by an electronic countermeasures receiver. Two main types of parallelism are employed in the creation of the parallel network. The first type is the grouping together of a large number of single data stream - multiple instruction stream processors under the supervision of one main control processor, and the second type involves data queueing via the the use of First-In-First-Out memories. Many tradeoffs are necessary in designing an architecture such as this and it is these tradeoffs as well as the overall system capability which are described in the following pages.

Architecture Overview

The first decision to be made is on how to steer the input data stream to the correct instruction stream. Also, the number of parallel instruction streams must be chosen. Here lies the first tradeoff. The number of instruction streams (which equates to the total amount of hardware) versus system performance must be balanced. If too few instruction streams are used, the system will not be able to handle the input data rate, and, if too many instruction streams are used, some of the hardware will be idle most of the time, and in airborne systems idle hardware is dead weight. To achieve optimum results this decision would require an extensive analysis

of ECM mission data and scenarios. Since this type of analysis would be classified and would be beyond the scope of this thesis an arbitrary decision will be made to test the feasibility of the parallelism concept. A network of 64 independent instruction streams and two levels of data stream steering logic will be used. On the first level, the azimuth coverage of the system will be divided into 16 sectors, and on the second level, four microprocessors will be assigned to each sector.

Data Stream Steering Logic

To achieve maximum speed, i.e., minimum delay in getting data to its proper destination, a logic array approach will be employed for data steering. One fairly stable parameter in a threat radar's pulse descriptor word (PDW) is angle of arrival (AOA). Even with the radio frequency receiver on an aircraft flying at near supersonic speeds, the rate at which the AOA of a particular radar's received pulses varies can be assumed to be zero with respect to a millisecond time interval. AOA is used as an address input to a logic array implemented in a random access memory. The data stored at each address is the proper destination for a pulse descriptor word with that AOA address. With AOA as the steering parameter, the azimuth coverage of the ECM system is first divided into 16 sectors of 22.5 degrees each. AOA is assumed to be an 8 bit quantity (based upon the B-1 RFS/ECMS system description) [REF 9]. The value of the least significant bit of this quantity would be 1.40625 degrees which is also the minimum angle resolution capability of the logic array. Figure 6 shows the composition of one of the input data words and Figure 7 is a block diagram of the first level of data steering logic.

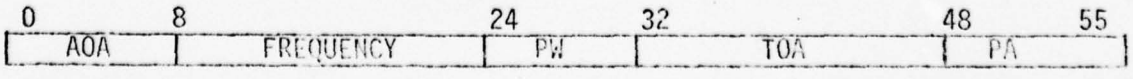


Figure 6.

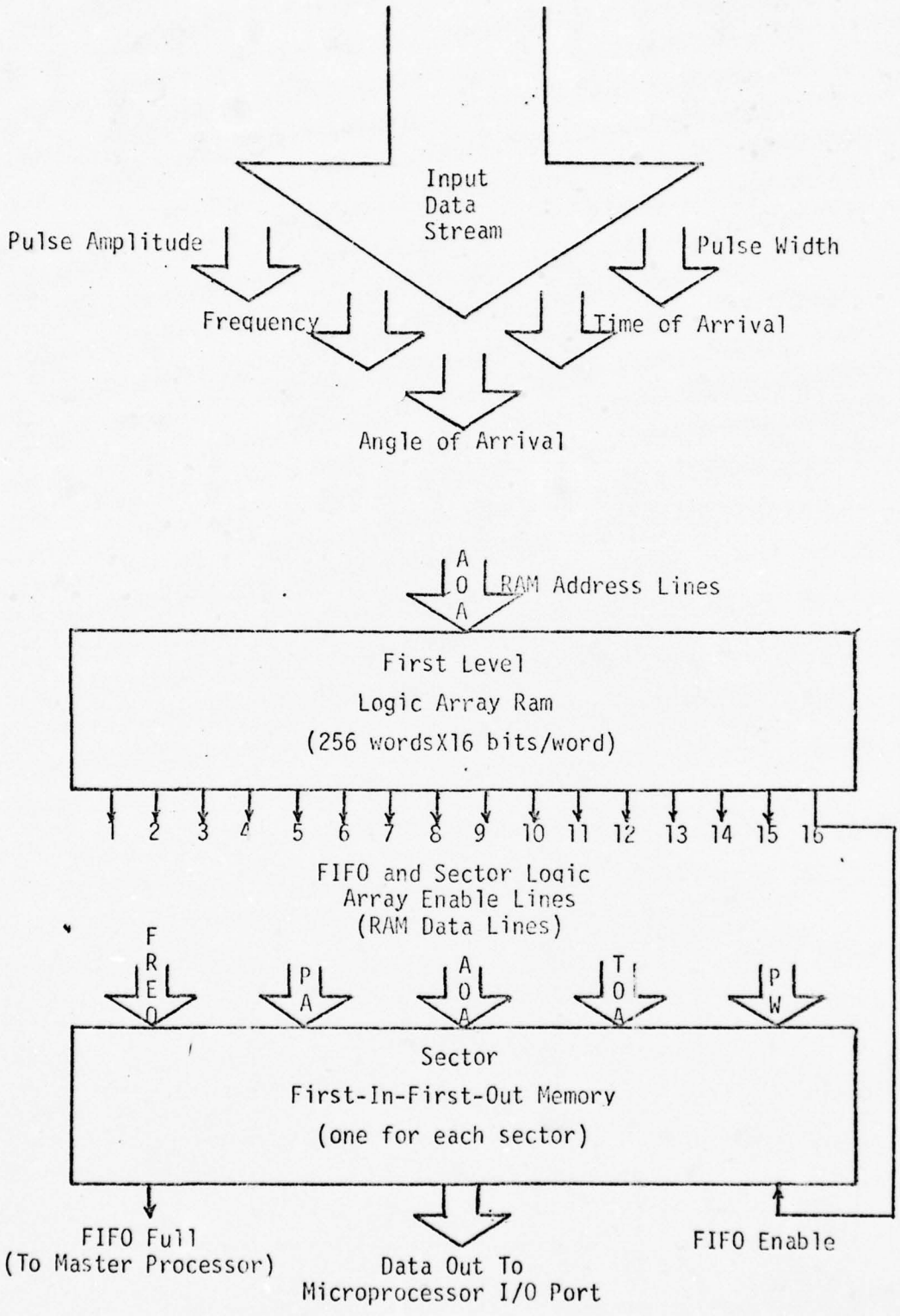


Figure 7. First Level Data Steering Logic

Initially, the first level logic array is loaded with 16 different sector addresses. Each sector's address is repeated a number of times according to its width in azimuth increments. The individual RAM addresses' contents are specified by the equation: $2^{(Sector-1)}$. The data stored at each address is a 16 bit quantity in which only one bit is set to a logical one. The position in the word of the bit which is set corresponds to a sector number. Fewer than 16 bits could be used to specify 16 possible data destinations but that number is the result of another tradeoff. Simplicity would dictate that only 4 bits be used but that would necessitate additional hardware. The logic array's data output is used for enabling the next level of logic and the sector FIFO's and if it were only a 4 bit quantity, it would require decoding. There is a tradeoff between additional hardware for decoding logic and a larger RAM word length to accomodate the longer data word length. Even with the longer word length, memory parts are available in the size required (4K bits) with acceptable access times (less than 100 nanoseconds). This results in an acceptably low contribution to the overall delay in getting a PDW to its ultimate destination.

As an enable signal, a RAM addresses' contents allows the PDW to be stored and held for the proper instruction stream to access. The next level of data steering logic is implemented in a similar fashion to the first level. Figure 8 depicts the second level of data steering logic.

The most obvious difference in the second level of logic is the fact that the logic array RAM is addressed by a 10 bit quantity consisting of all 8 bits of AOA and the 2 most significant bits of frequency. The incorporation of frequency in the RAM address at this point allows flexibility in choosing the PDW's final destination. This flexibility is used

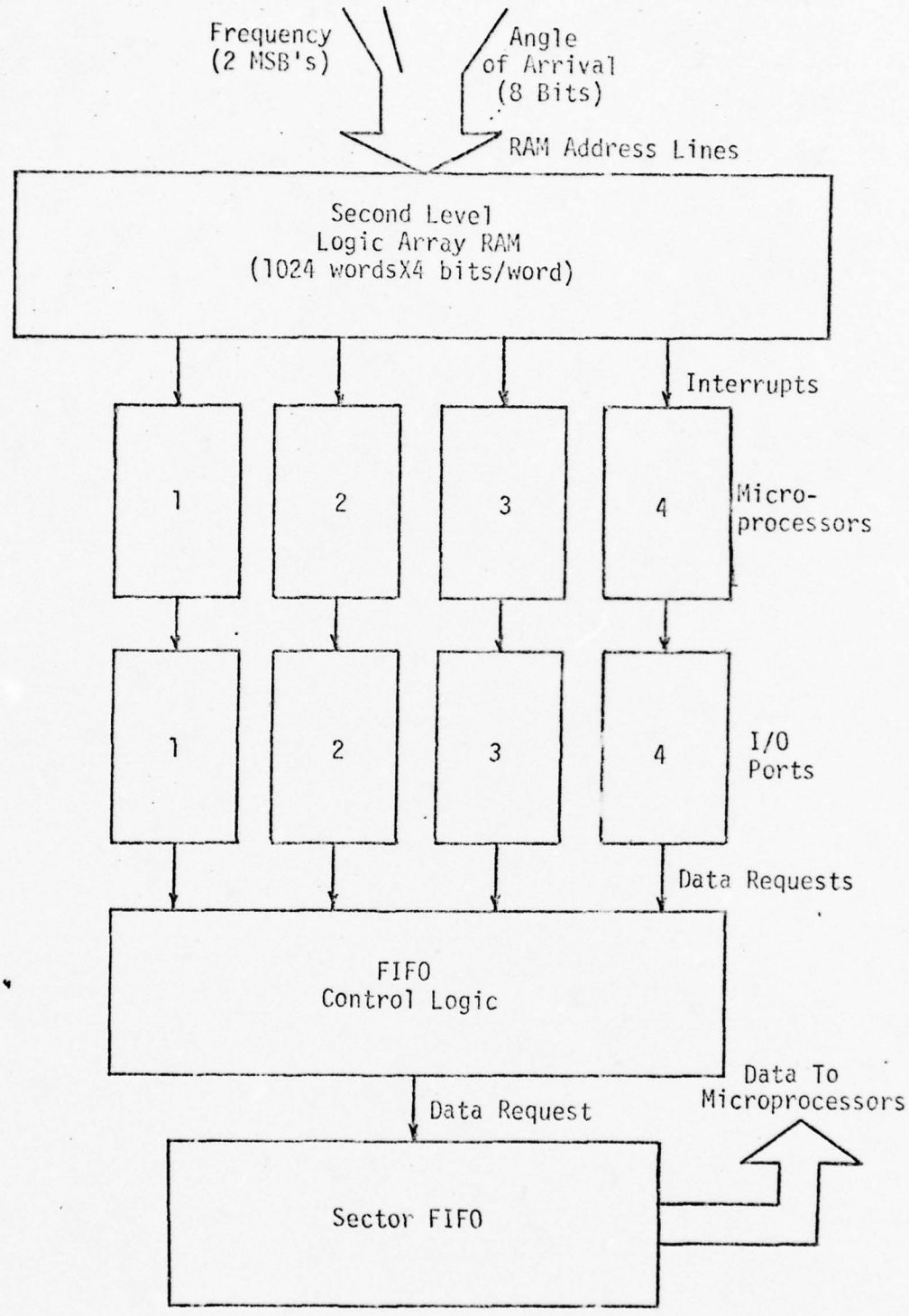


Figure 8. One Sector's Data Steering Logic

to help handle the situation where one sector is overloaded with a disproportionate amount of data compared to the others. The overall system is designed around an average data rate per sector and overload conditions can result when there are a large number of radar pulses being received from one azimuth sector. Overloads are discussed beginning on page 31.

Each of the sector logic array RAM's is 4K bits long and is arranged into 1024 four bit words. As in the first level logic array, the 4 bits of data in each address correspond to data destinations, however, in this case it is the data's final destination. The RAM's contents are specified by the equation: 2^{bin}, where bin represents one of the four microprocessors in a particular sector. Also similar to the first logic level is the use of four bits where two would serve, to eliminate the need for decode logic.

One of the four processors (instruction streams) is informed of its selection via interrupt from the logic array. Once an instruction stream is chosen, data is fed into it from the FIFO. The microprocessor accesses the FIFO via an input/output port and extracts the data on which it must operate. One more tradeoff is encountered here. If only one FIFO per sector is used it must be connected to some control logic to queue the microprocessor's requests for data and insure that the data gets to the correct processor. If instead, one FIFO per processor is used the control logic can be eliminated. One FIFO per sector is assumed and as with the first level logic array, accessing the RAM contributes less than 100 nanoseconds to the total delay time of the system.

The final delay in getting data to its proper location is in accessing the FIFO. This delay is also less than 100 nanoseconds, based upon the memory parts currently available; therefore, the total delay adds up to approximately 250 nanoseconds. Input time for the microprocessors is not

considered becuase it is not a delay caused specifically by system choices, i.e., an input operation would be required ragardless of the system architecture. Therefore, the total delay time from receipt of a pulse to the start of processing on that pulse is compatible with the requirements of the most capable Air Force ECM system, the B-1 RFS/ECMS [REF 9].

System Overloads

To cope with the situation where too much data is flowing into a sector for it to handle, i.e., the FIFO fills up and overflows, the partitioning of the 360 degrees of azimuth into 16 sectors must be adaptable. First, however, the overload condition must be detected. This is accomplished by using the FIFO full signal from each of the 16 sector FIFO's as a prioritized interrupt to a master system monitoring processor as shown in Figure 9. Prioritization of the interrupts can be used to place

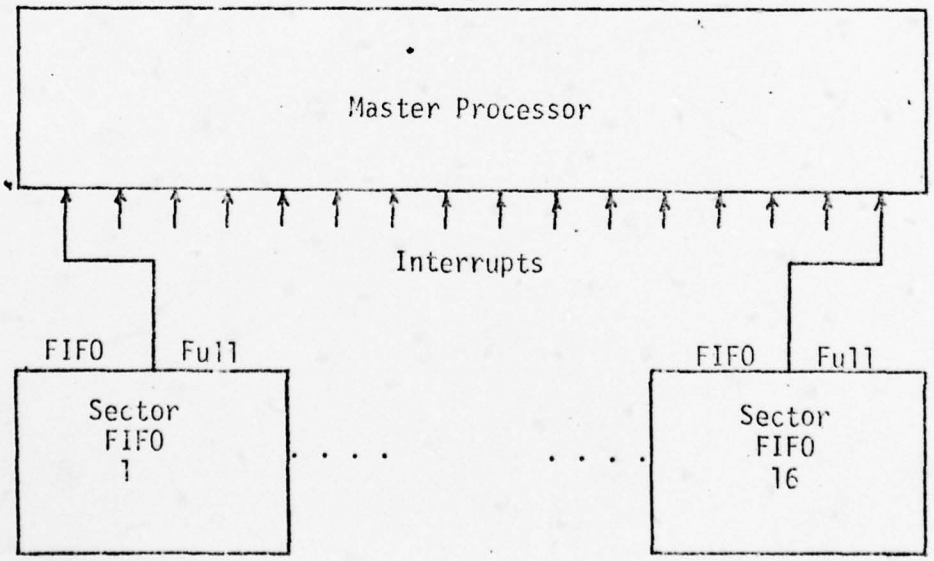


Figure 9. The Master Processor

increased emphasis on some sectors to deal with simultaneous overloads.

Basically what must be accomplished by the master processor is a reformatting of the logic array stored in the first level RAM and the logic array in the overloaded sector. When a sector is receiving too much data that load can be reduced by narrowing the azimuth range the sector must cover. Narrowing one sector's azimuth coverage means that the coverage of another must be expanded, given a fixed number of sectors. Both logic arrays involved must be reformatted but there is a minimum azimuth beyond which a bin should not be shrunk. This minimum is related to the azimuth measurement accuracy of the ECM system. If a minimum sector width of twice the azimuth accuracy of the ECM system is used, the data within any given sector should be stable long enough for processing to be accomplished. If a smaller sector width is used the probability that one pulse from a threat emitter will be in one sector and the next will appear in another sector, before processing is completed on the pulse train, becomes much greater. The operation of the master processor's software programs is highlighted in the following chapter.

VI Software

Software programs to detect and service overloads, reduce overload conditions, and reformat logic array contents are documented in following sections. Basically, these programs are of two types: those which would normally reside in the master processor and those which are included for simulation purposes. The interrupts are simulated, input data is fed into the processor network from a table, and the individual pulse descriptor words are stored in arrays instead of FIFO's.

Normal Processing Software-Subroutine NORPROS

Functional Description. This is the software program which would be the executive in the master processor. An overload occurs when any sector FIFO becomes full and can no longer accept new data and results in the narrowing of a sector's azimuth coverage. It would be executed continuously in a loop as long as there were no overloads. Its main purpose is to return overloaded sectors to their normal state. There is another system tradeoff here. It is desirable to attempt overload reduction as soon as possible but there is an inherent danger here in that overload reduction too soon could cause a sector to re-overload and hence cause an excessive amount of time to be spend in reformatting arrays. While a sector is overloaded it cannot process pulse descriptor words until its logic array has been reformatted, i.e., it cannot perform its main function. Therefore, attempts at overload reduction too soon can result in too many pulses being missed from a specific emitter's pulse train.

Variables. Following is a list of the variables used within the subroutine and a definition of their purposes.

A. TIMES-This is a "simulation" variable which determines how many times the subroutine is executed before it is interrupted by a simulated overload. Normally, the subroutine would execute continuously in a loop until interrupted.

B. OVLTAB-This is the main overload table and is used by all subroutines. It is a 16 X 4 array, one entry for each sector. Within each entry there are four variables.

1. OVLTAB(SECTOR,1) is a flag called OVLFLAG which when set to one indicates that a sector is overloaded.

2. OVLTAB(SECTOR,2) is a variable called OVLSTAT which indicates the status of a sector's azimuth coverage.

a. OVLTAB(SECTOR,2)=0 means that a sector is expanded to 28.125 degrees, i.e., it is covering some of the azimuth spectrum left uncovered when an overloaded sector was shrunk.

b. OVLTAB(SECTOR,2)=1 indicates normal azimuth coverage by a sector.

c. OVLTAB(SECTOR,2)=2 means that a sector has overloaded once and now covers 16.875 degrees instead of its original 22.5 degrees.

d. OVLTAB(SECTOR,2)=3 indicates that a sector is double overloaded and is currently covering 11.25 degrees of azimuth. It requires two other sectors to be expanded.

e. OVLTAB(SECTOR,2)=4 means that a sector is at its narrowest azimuth coverage of 5.625 degrees. In this state the frequency spectrum coverage is divided in half so that two of the sector's micro-

processors cover each half of the frequency spectrum and all of the narrow azimuth sector. This is done to provide partitioning of the data stream in this sector via the frequency parameter when AOA can no longer be used.

f. OVLTAB(SECTOR,2)=5 is the most severe overload condition. In this state frequency coverage is decreased. A sector is only 5.625 degrees wide and the microprocessors assigned to it cover only portions of the frequency spectrum, i.e, pulses which are received in this sector's azimuth range but which are considered unimportant (via their frequencies) are not processed.

3. OVLTAB(SECTOR,3) is a pointer called EXPANDR which points to a sector which caused another sector to be expanded. This pointer is only valid when OVLSTAT=0 and should otherwise be zero.

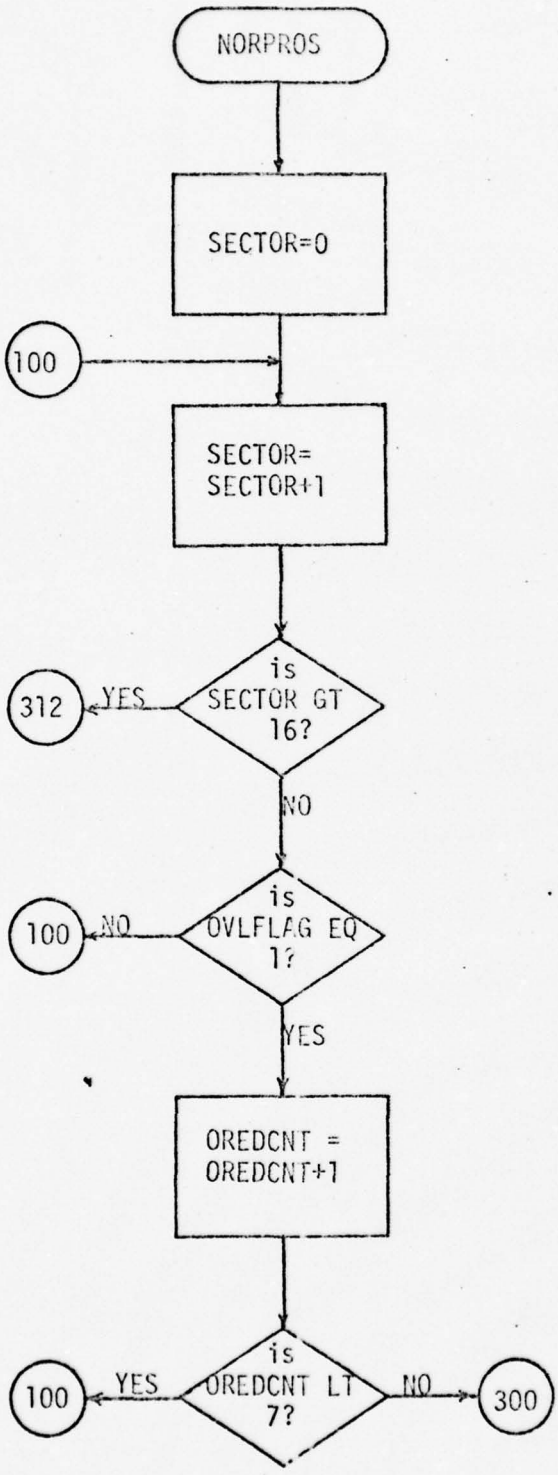
4. OVLTAB(SECTOR,4) is a counter called OREDCNT and is used to keep track of how long a sector has been in its current overload state. For test purposes, the count is allowed to reach seven before an attempt is made to reduce a sector's overload status, i.e., NORPROS is executed seven times. Seven is an arbitrary number and in an actual system a much larger number would probably be used since its maximum value is determined by the actual time to execute NORPROS. During the period of time defined by the maximum value allowed for OREDCNT (and NORPROS's execution time) there should at least be a possibility that the overload has vanished, otherwise the sector will re-overload. The test scenario in Appendix C illustrates this condition.

C. SECTOR-This variable is used primarily as a table search pointer when searching OVLTAB for a particular condition or parameter.

D. CUROVL-When an overloaded sector is found, CUROVL is set

equal to the sector's number. This variable is used primarily by subroutine FSTRAM as an indicator of the sector whose azimuth coverage is to be reduced.

Flowcharts. Figure 10 shows the detailed logic flow of this subroutine. A commented listing of its implementation in FORTRAN IV can be found in Appendix A.



Initialize the sector counter.

Increment the sector counter.

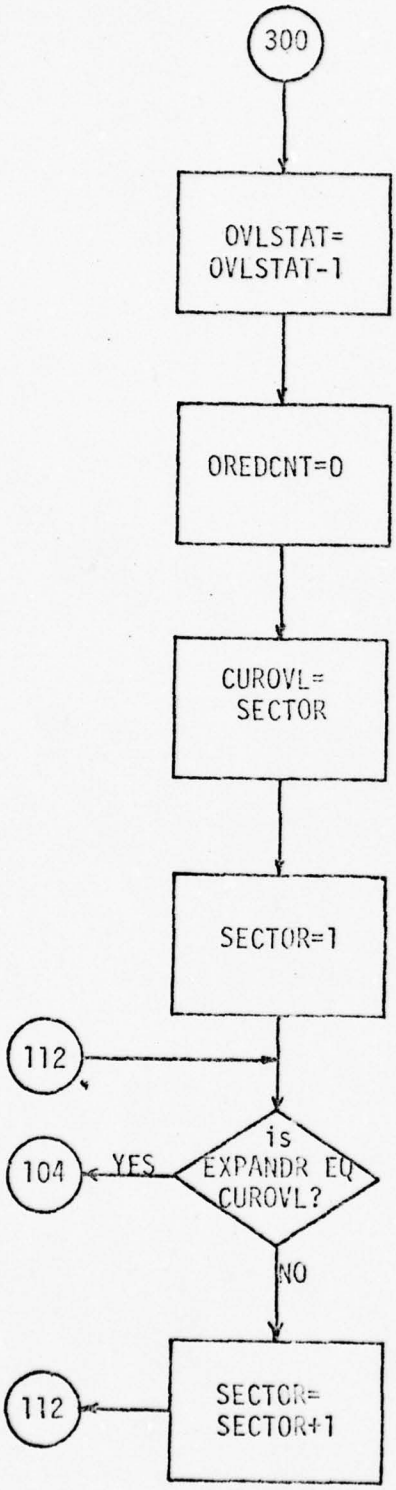
Have all sectors been checked?

Is this sector overloaded?

Increment the overload reduction count.

Is it time to attempt a reduction in this sector's overload status?

Figure 10. Subroutine NORPROS



Decrement this sector's overload status.

Reset the overload reduction counter.

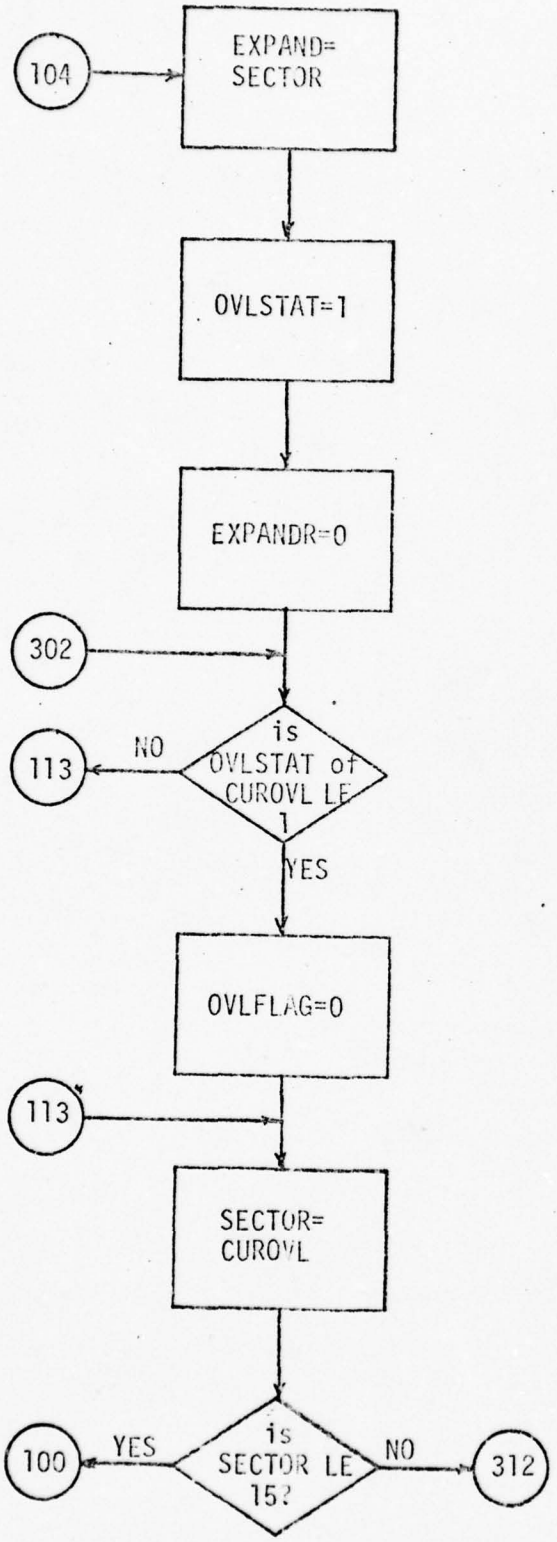
Set the variable CUROVL to this sector's number so that its azimuth coverage can be adjusted according to its new overload status by FSTRAM.

Re-initialize SECTOR for the search for the sector which was expanded previously by CUROVL.

Is this the correct sector?

Increment the sector counter.

Figure 10(cont).



Set the variable EXPAND to this sector's number so that its azimuth coverage can be returned to normal by subroutine FSTRAM.

Reset the overload status of this sector from expanded back to normal.

Reset this sector's EXPANDR pointer.

Has the overloaded sector's overload status been reduced all the way back to normal?

Reset its overflag.

Restore the sector counter back to the currently overloaded sector's number so that the search for more overloaded sectors can continue.

If SECTOR is less than or equal to 15 than there are more sectors to check.

Figure 10(cont).

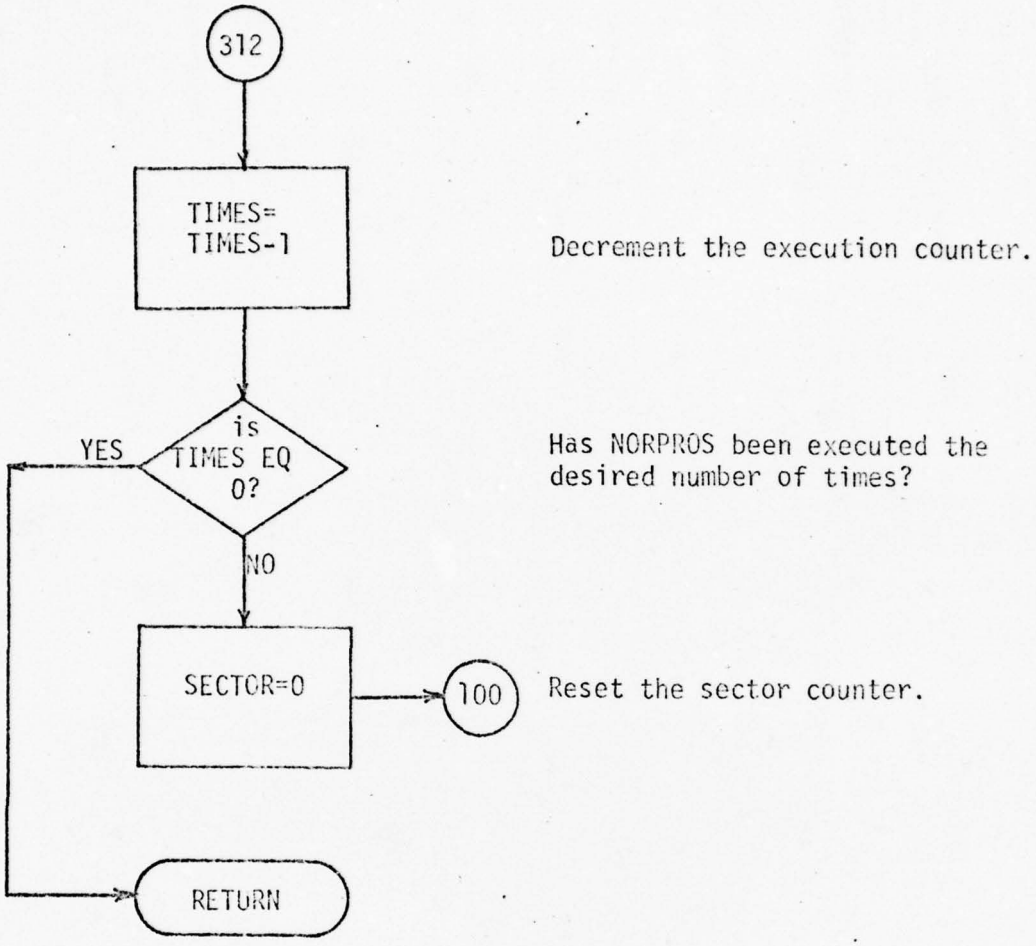


Figure 10(cont).

Interrupt Processing Software - Subroutine INTSERV

Functional Description. The subroutine INTSERV is partially simulation and partially an interrupt service routine. It contains logic that simulates the occurance of a hardware interrupt to the master processor, as well as logic to service that interrupt. Normally the interrupts would occur as the sector FIFO's became overloaded. This subroutine's basic function is to determine which sector is overloaded, find a sector which can be expanded, and call the subroutine FSTRAM so that the system's logic arrays can be reformatted.

Variables. The variables used by the subroutine INTSERV, their definitions, and purposes are as follows:

A. OVLSECT - This is a "simulation" variable. It supplies subroutine INTSERV with the overloaded sector's number. In a real system the overloaded sector's number would be determined via the master processor's interrupt logic.

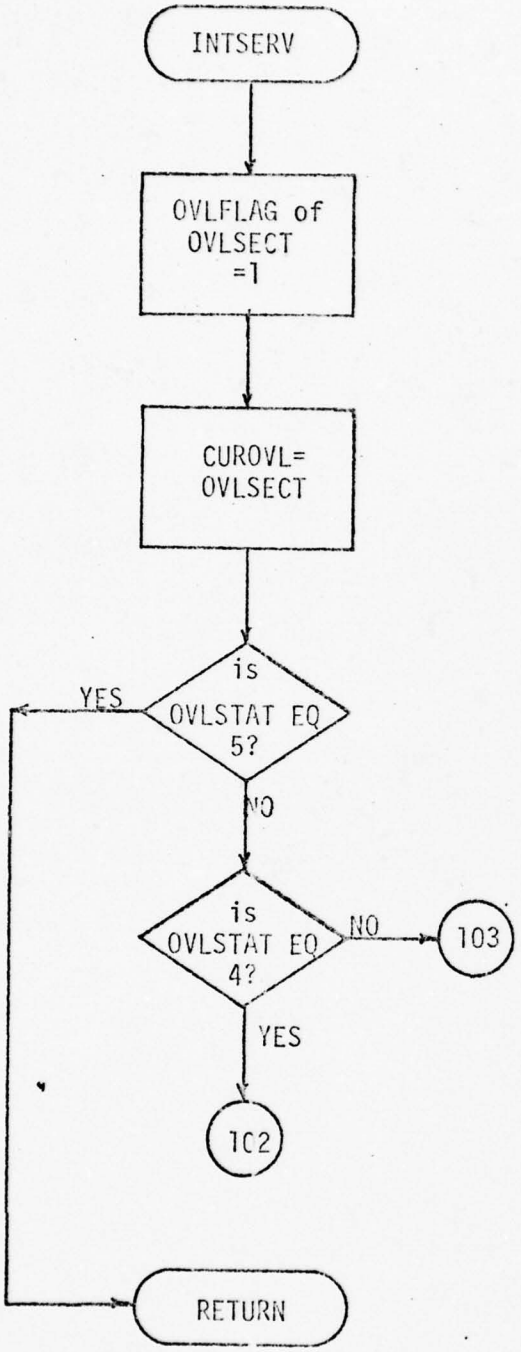
B. OVLTAB - This is the same table as used by NORPROS.

C. CUROVL - This the same variable as used by NORPROS, but in INTSERV it represents the sector which is actually overloaded.

D. SECTOR - This is the same pointer as used by NORPROS.

E. EXPAND - This is the same variable as used in NORPROS, but in INTSERV it represents the sector which is actually to be expanded.

Flowcharts. Figure 11 shows the detailed logic flow of subroutine INTSERV. A commented listing of its creation in FORTRAN IV can be found in Appendix A.



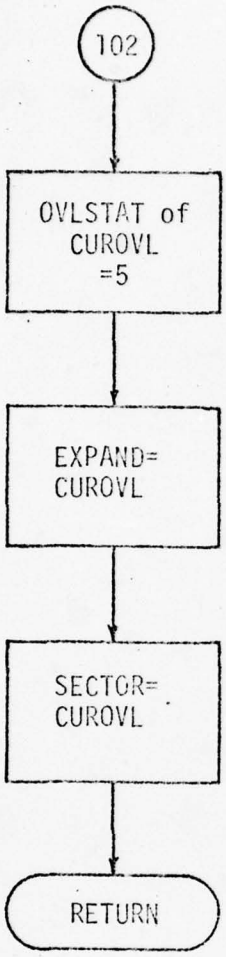
Set the overload flag for the overloaded sector.

Set the current overload variable to the number of the sector which is overloaded.

Check the overloaded sector's overload status. If it is already in overload state 5 then nothing else can be done.

If OVLSTAT is now equal to 4 it will be incremented to 5 (at 102) which is a frequency stripping state. Otherwise, its azimuth coverage will be modified in accordance with its new overload state.

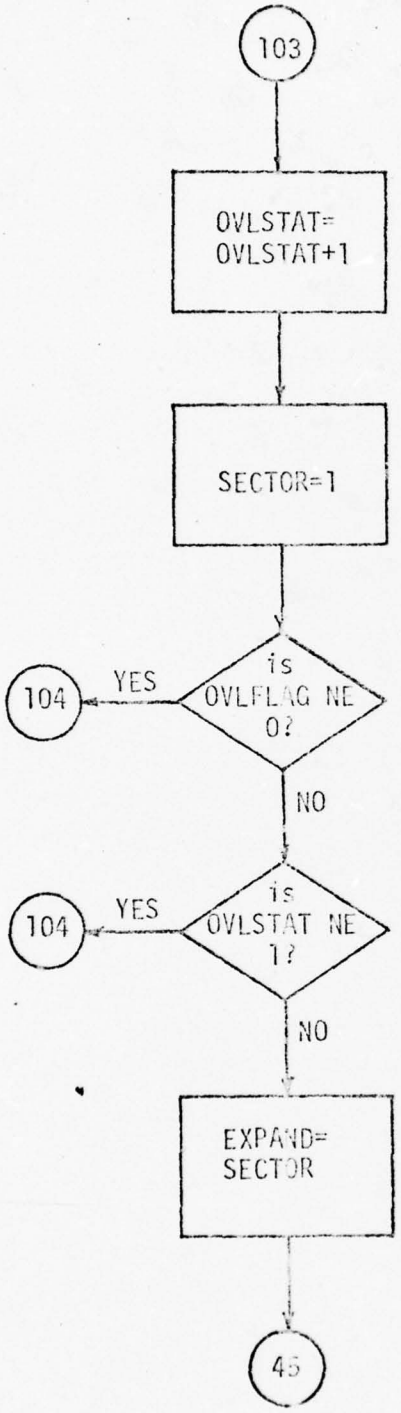
Figure 11. Subroutine INTSERV



Set this sector's overload state to 5.

Set both the sector search counter and the expand variable to the number of the sector which is overloaded. EXPAND will be used by FSTRAM to set up frequency stripping for that sector and SECTOR is also set to that value to act as a pointer to this sector's entry in the overload table.

Figure 11 (cont).



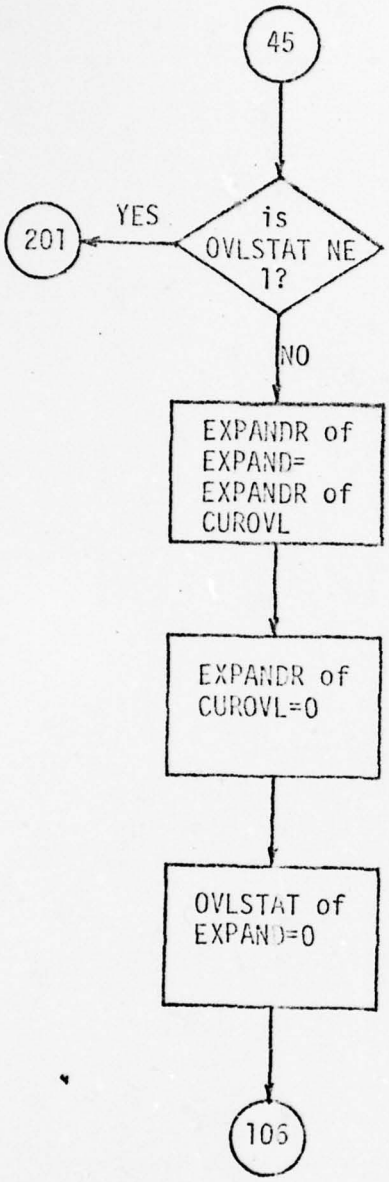
Increment this sector's overload status.

Initialize the sector search counter for the search for a sector which can be expanded.

If OVLFLAG is equal to zero, i.e., this test is failed, then the sector in question is not overloaded but it must be checked to see if it is already expanded. It is expanded if OVLSTAT=0.

Set EXPAND equal to SECTOR so that FSTRAM will expand the sector which was found to be available.

Figure 11 (cont).



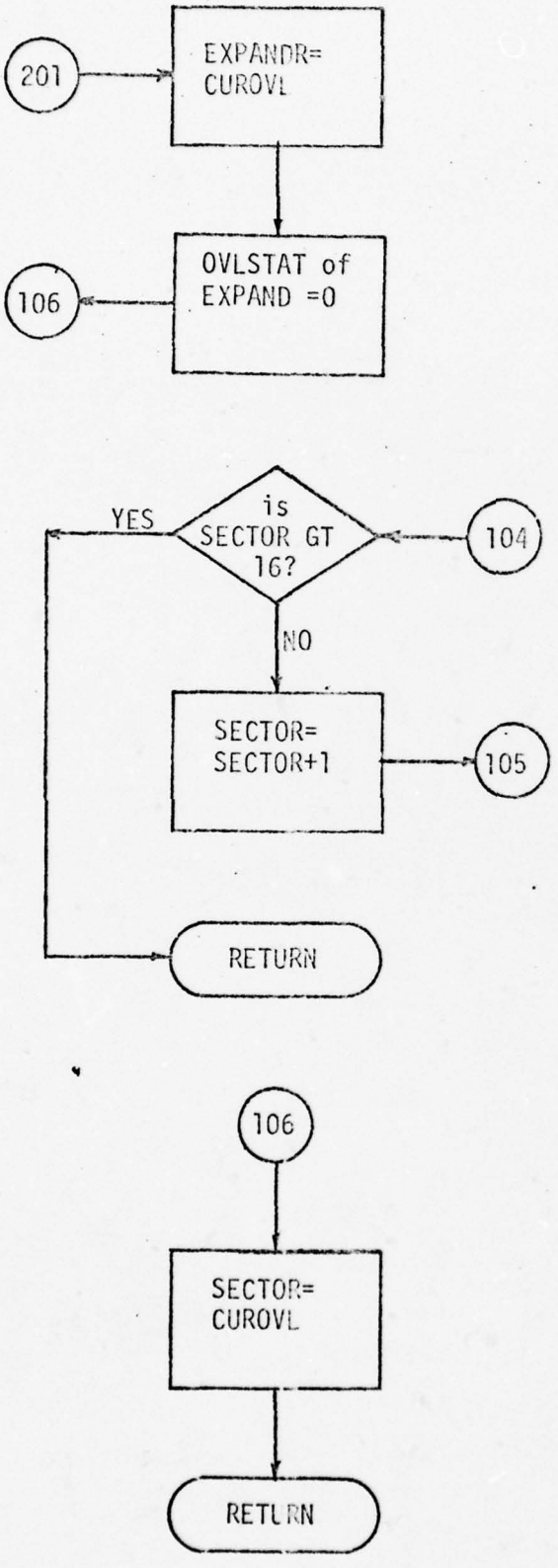
Test to see if the currently overloaded sector is one which used to be expanded.

Pass the EXPANDR pointer from the now overloaded sector (which used to be expanded) to the new sector which was found to take its place.

Reset the currently overloaded sector's EXPANDR pointer to zero.

Set the new expanded sector's overload state to zero indicating that it is now expanded.

Figure 11 (cont).



Set the expandable sector's EXPANDR pointer to the overloaded sector's number.

Set the expanded sector's overload state to zero.

Have all sectors been searched for one which can be expanded? If so, ther nothing can be done. If not, then increment the sector search pointer and try again.

Restore the sector pointer to the number of the sector which was overloaded.

Figure 11 (cont).

First Level Logic Array Subroutine - FSTRAM

Functional Description. The subroutine FSTRAM does the reformatting of the first level logic array program. It is called from subroutine INTSERV to shrink the azimuth coverage of an overloaded sector and expand another sector to pick up the slack. FSTRAM is also called from NORPROS to reformat the logic array when a sector's overload status is to be reduced. In the latter case, FSTRAM expands the azimuth coverage of the overloaded sector in accordance with its new overload state, and shrinks the sector which the overloaded one had expanded. In neither case is the entire array reformatted unless sector 1 is overloaded and sector 16 is to be expanded or vice versa. This is done to minimize the time spent between the receipt of an overload interrupt by NORPROS and the actual return to normal processing with the new logic array program. The program listing in Appendix C provides additional details on FSTRAM's operation.

Variables. The following variables are used, created, or modified by FSTRAM.

A. EXPAND is a variable indicating the sector to be expanded if set by INTSERV or the sector which is having its overload state reduced (azimuth coverage expanded) if set by NORPROS.

B. CUROVL is a variable indicating the sector which is currently overloaded.

C. RAMENT is a variable which points to the logic array element being modified.

D. WIDTH is a variable indicating the width of a sector in azimuth increments (or logic array elements). It varies according to a sector's overload state.

E. SECTOR is a general purpose table search pointer.

F. RAMTAB is a table which contains the new logic array program.

G. OVLTAB is a general purpose table indicating the overall condition of the system.

H. SRAMTAB is the table which contains each sector's logic array program. It is not used by FSTRAM.

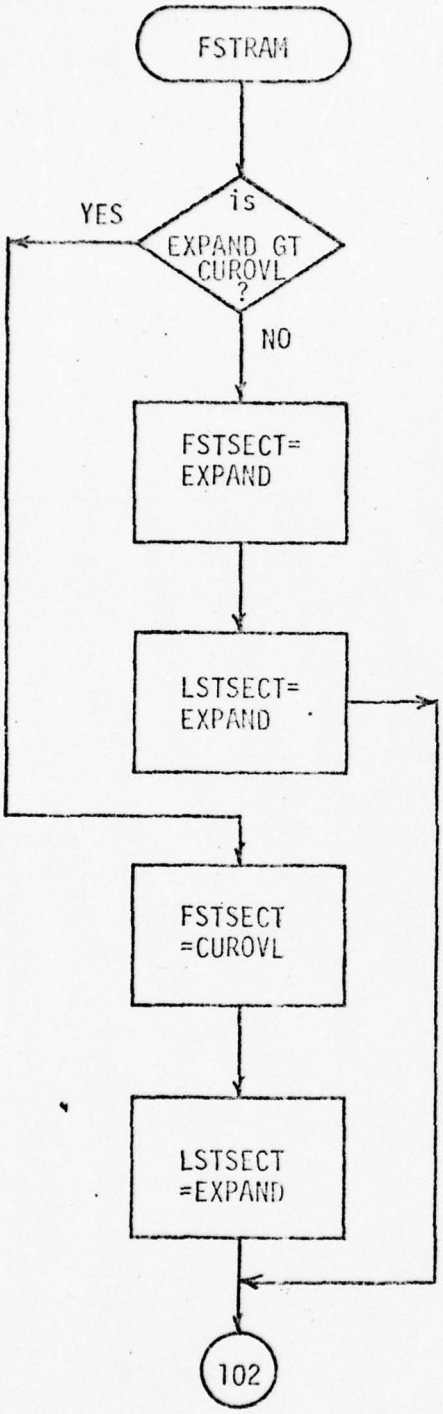
I. FSTSECT is a variable indicating the first sector to be reformatted.

J. LSTSECT is a variable indicating the last sector to be reformatted.

K. BINCONT is a variable which is calculated by the equation: $BINCONT=2^{(SECTOR-1)}$, and is stored in RAMTAB.

L. LSTENT is a variable indicating the last logic array element within a sector to be modified.

Flowcharts. Figure 12 shows the detailed logic flow for subroutine FSTRAM. A commented FORTRAN listing is contained in Appendix A.

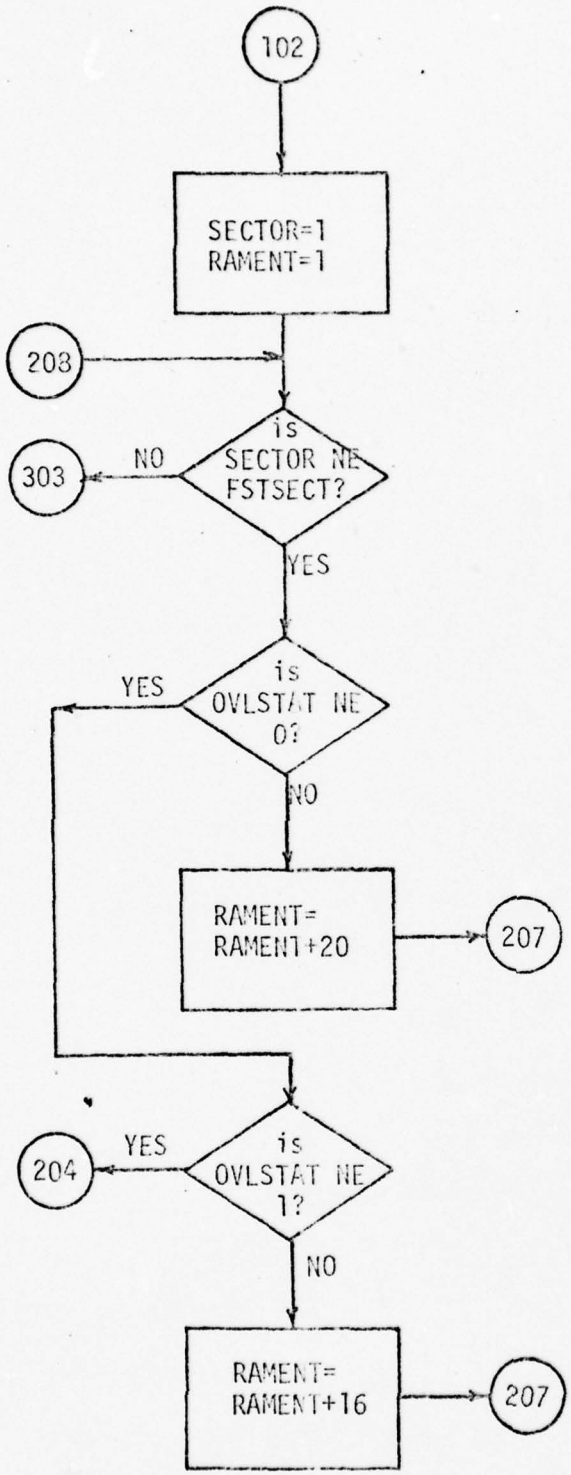


Which sector comes first, the one to be expanded or the one which is overloaded?

If the expanded sector comes first in OVLTAB then set FSTSECT equal to its number and set LSTSECT equal to the number of the sector which is overloaded.

If the overloaded sector comes first in OVLTAB then set FSTSECT equal to its number and set LSTSECT equal to the number of the sector which is to be expanded.

Figure 12. Subroutine FSTRAM



Initialize the variables SECTOR and RAMENT for the calculation of the cumulative number of azimuth increments in the sectors preceeding FSTSECT.

Has the first sector to be reformatted been reached?

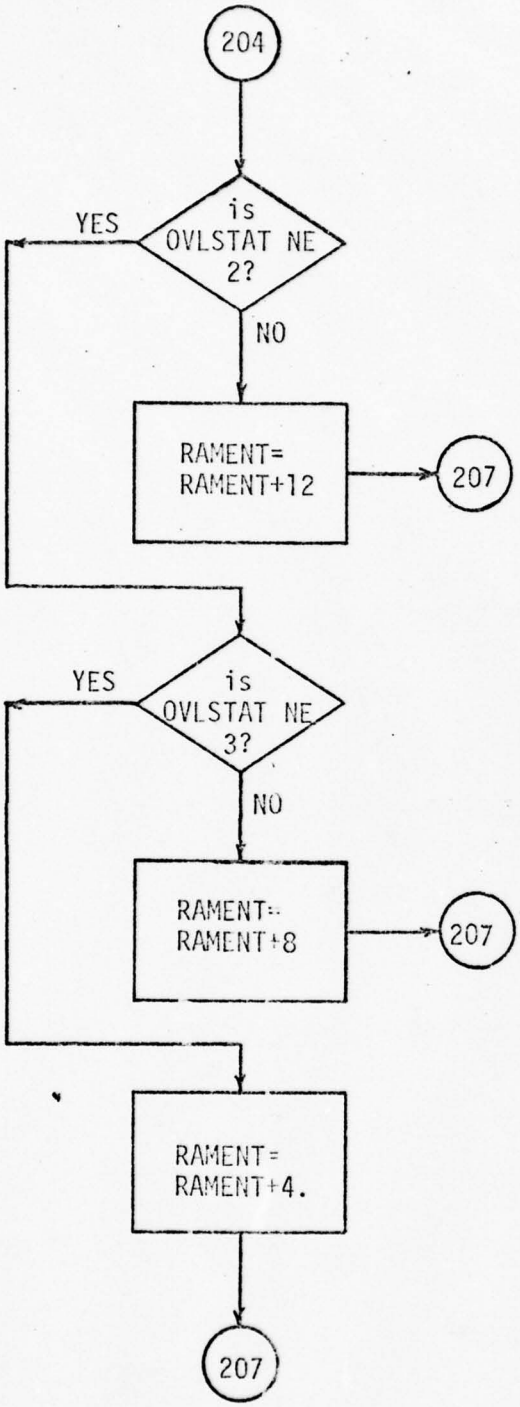
If this sector's overload status is zero then its width is 20 azimuth increments.

Increment RAMENT by 20.

If this sector's overload status is one then its width is 16 azimuth increments.

Increment RAMENT by 16.

Figure 12 (cont).



If this sector's overload status is two then its width is 12 azimuth increments.

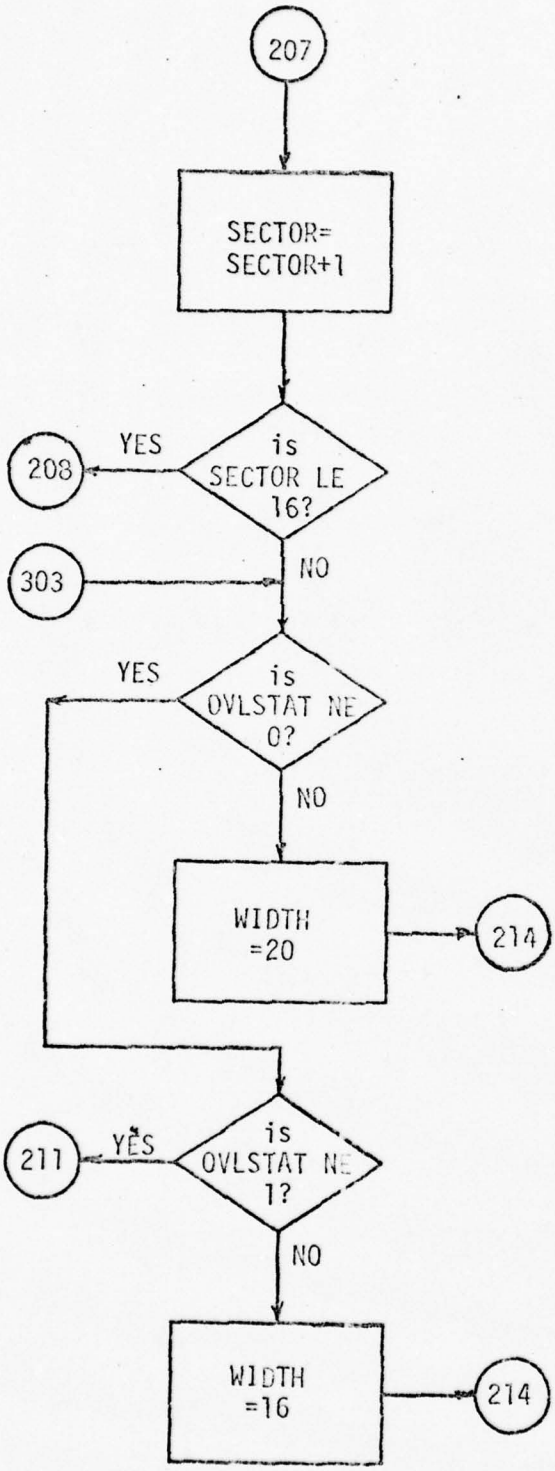
Increment RAMENT by 12.

If this sector's overload status is three then its width is 8 azimuth increments.

Increment RAMENT by 8.

The sector's overload status must be 4 or 5, therefore, its width is 4 azimuth increments. Increment RAMENT by 4.

Figure 12 (cont).



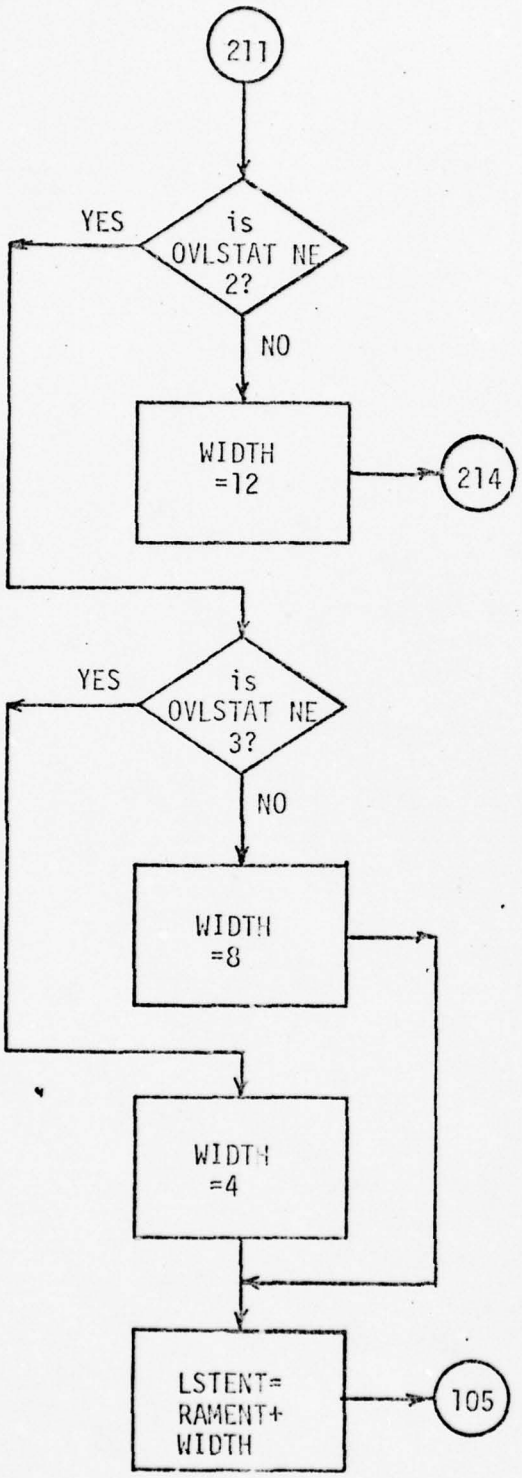
Increment sector.

Has the end of the overload table been reached? If not, continue to calculate RAMENT. If so, find the width of the sector being reformatted.

If OVLSTAT=0 the sector's width is twenty.

If OVLSTAT=1 the sector's width is sixteen.

Figure 12 (cont).



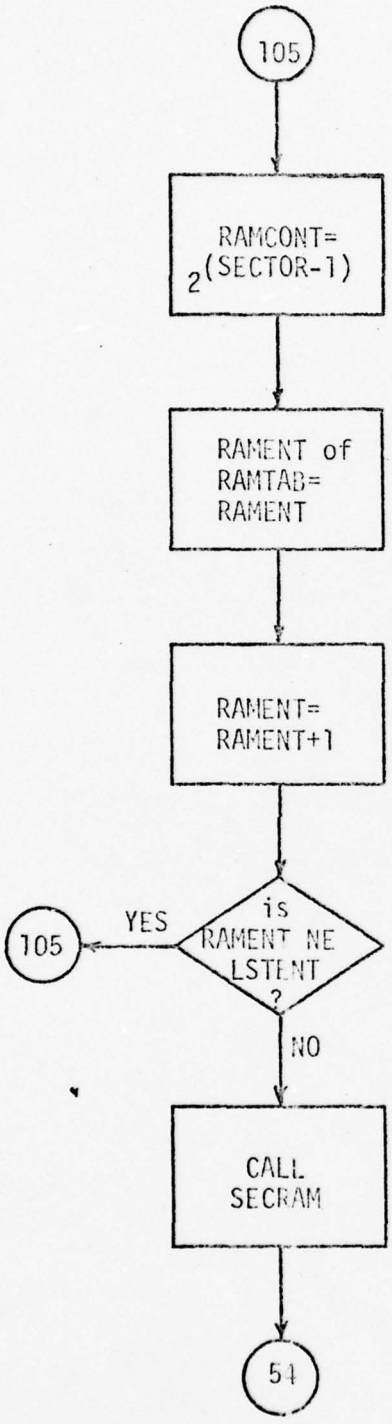
If OVLSTAT=2 then the sector's width is twelve.

If OVLSTAT=3 then the sector's width is eight.

If OVLSTAT=4 or 5 then the sector's width is four.

LSTENT is the last logic array element to have this sector's number stored in it.

Figure 12 (cont).



Calculate the logic array element's contents.

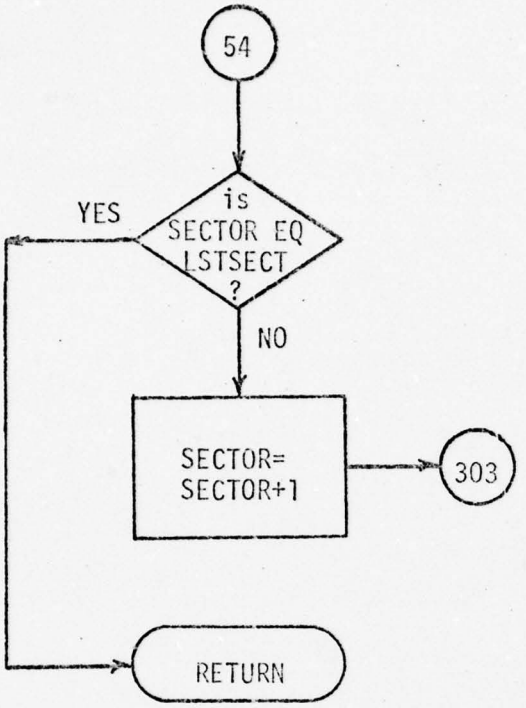
Store the sector's number in the logic array element specified by RAMENT.

Increment RAMENT.

Have all logic array elements which are assigned to this sector been modified?

Call the subroutine SECRAM so that it can reformat the sector's logic array.

Figure 12 (cont).



Have all sector's which are supposed to be reformatted been done?

Increment SECTOR.

Figure 12 (cont).

Second Level Logic Array Subroutine - SECRAM

Functional Description. The subroutine SECRAM is responsible for the reformatting of the logic arrays which perform the second level (sector level) of data steering. It is called by the subroutine FSTRAM each time FSTRAM completes its processing on a particular sector. Each azimuth sector has four microprocessors available for assignment and the sector logic arrays steer the incoming data to the proper processor bin.

When a sector's azimuth coverage is modified the assignments of the microprocessors within that sector must also be modified. If a sector's coverage is narrowed to less than 12 azimuth increments (16.8 degrees), frequency is used in addition to AOA as a steering parameter. The two most significant bits of the frequency parameter are concatenated with the 8 bits of AOA forming a 10 bit addressing parameter for use with the second level logic arrays. The methodology for storing the data destinations in their appropriate addresses is as follows. If a sector's width is 12 azimuth increments or greater, the destinations (bin assignments) are stored in four groups of locations specified by the addresses: 00AAAAAAAA, 01AAAAAAAA, 10AAAAAAAA, and 11AAAAAAAA. The first two bits are the frequency bits and the others are the 8 bits of AOA. In essence, the frequency bits in the address are ignored. If a sector's width is 8, one frequency bit is used and the destinations will be found at the addresses 0FAAAAAAAA and 1FAAAAAAAA. Finally, if a sector's width is 4, both frequency bits are used and the destinations can be found at the addresses specified by FFAAAAAAAA.

Subroutine SECRAM stores the new logic array elements in a table called SRAMTAB and returns control to FSTRAM. Like FSTRAM, it only reformats the array elements which must be modified thus minimizing its

execution time. A program listing for SECRAM's implementation in FORTRAN IV is contained in Appendic C.

Variables. The following variables are used, created, or modified by SECRAM.

A. SRAMENT is a variable similar to RAMENT in that it points to a specific logic array element in the array table called SRAMTAB.

B. SRAMTAB is the sector logic array table.

C. BWIDTH is a variable which indicates the size of a single bin within a sector when a sector's width is at least 12 azimuth increments.

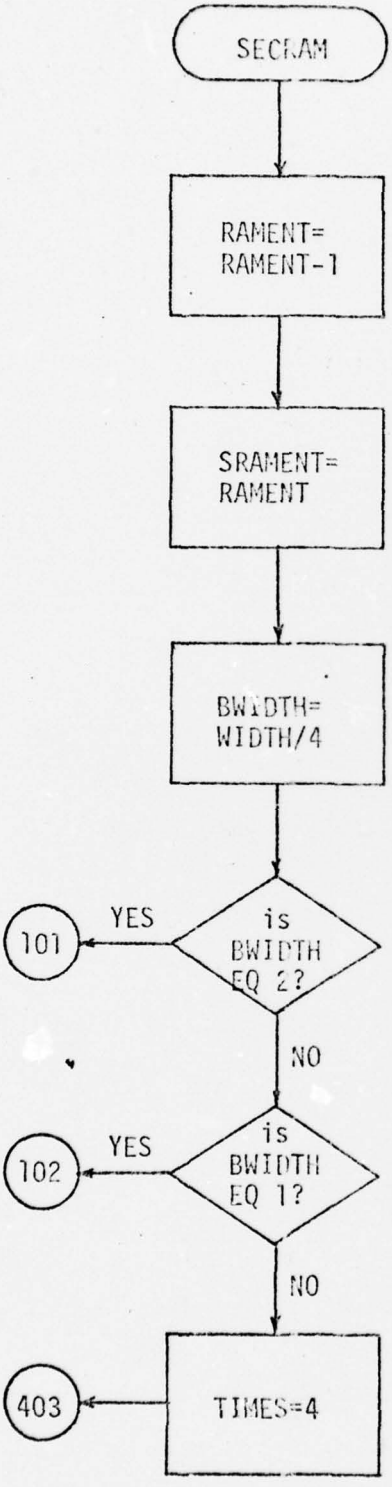
D. TIMES is a counter which keeps track of how many times a loop in SECRAM has been executed.

E. BIN is a variable equal to a microprocessor's bin number.

F. BINCONT is a variable which indicates the desired destination for the data entering a sector. BINCONT is calculated by the equation: $BINCONT=2^{BIN}$.

G. WIDCNT is a variable indicating the final logic array element to be modified during a particular execution of SECRAM.

Flowcharts. Figure 13 shows the detailed logic flow for subroutine SECRAM. A commented listing is contained in Appendix A.



Decrement RAMENT to the last first level logic array element to be modified. SECRAM will modify SRAMTAB's elements in decending order from that point.

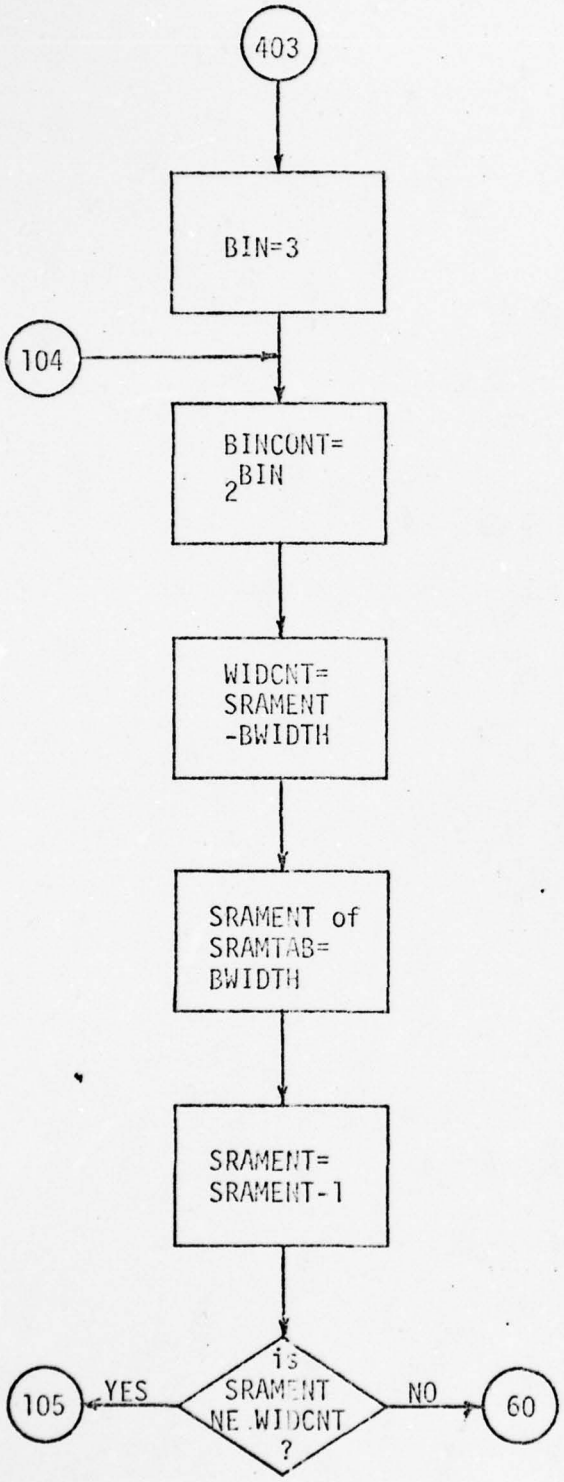
SRAMENT is SECRAM's pointer into its logic array.

For all sectors which are at least 12 azimuth increments wide, each microprocessor covers 1/4 of the assigned azimuth.

The next two decisions check to see if the sector's width is less than 12 azimuth increments.

If a sector's width is 12 or greater then the data destinations must be stored in 4 sets of locations to effectively ignore the frequency portion of the RAM address.

Figure 13. Subroutine SECRAM



Start with bin number three.

Calculate BINCONT.

Calculate the number of the last logic array element to be modified.

Store the data destination BINCONT in its proper location in SRAMTAB.

Decrement SRAMENT to the next element to be modified.

Have all the logic array elements been modified?

Figure 13 (cont).

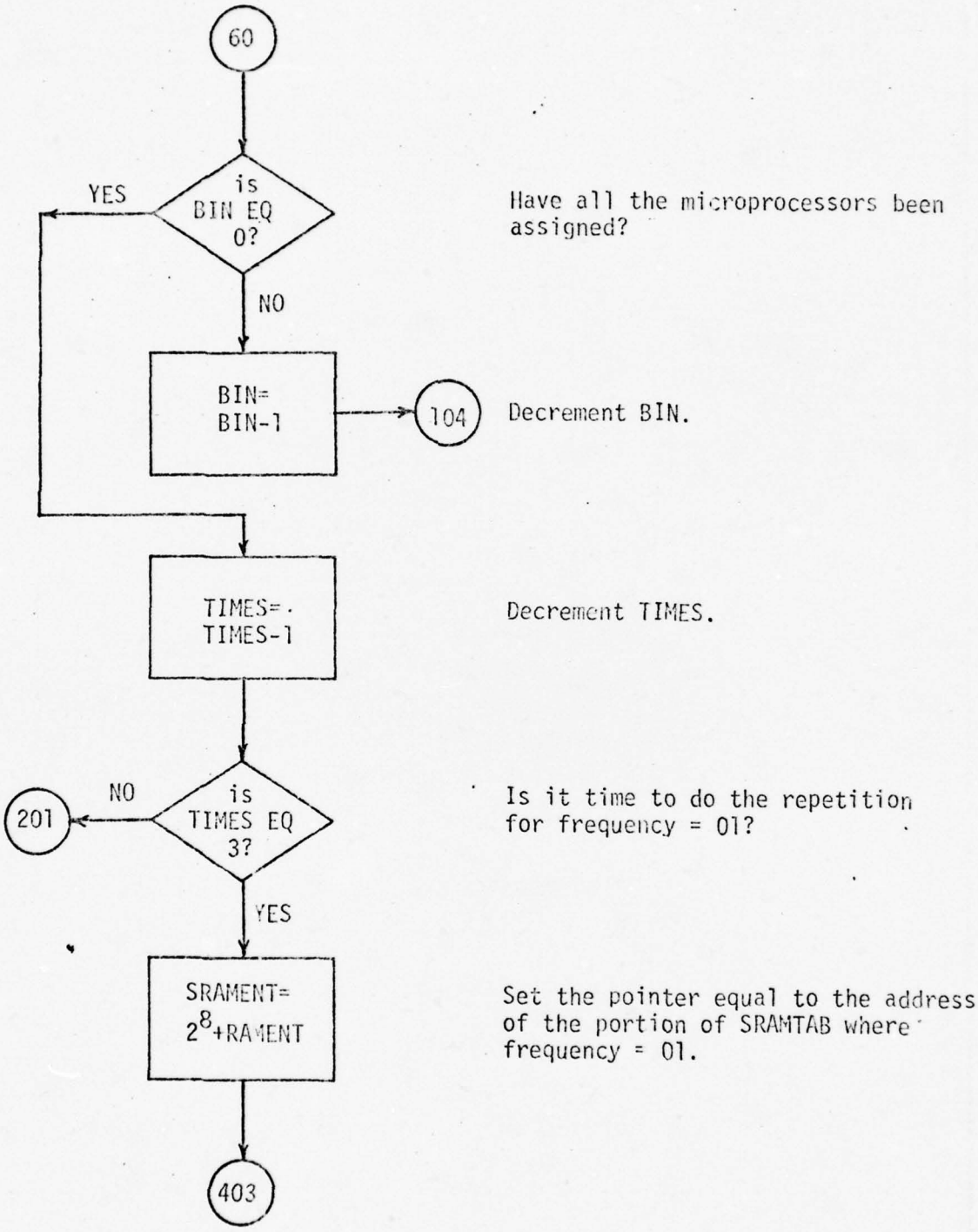


Figure 13 (cont).

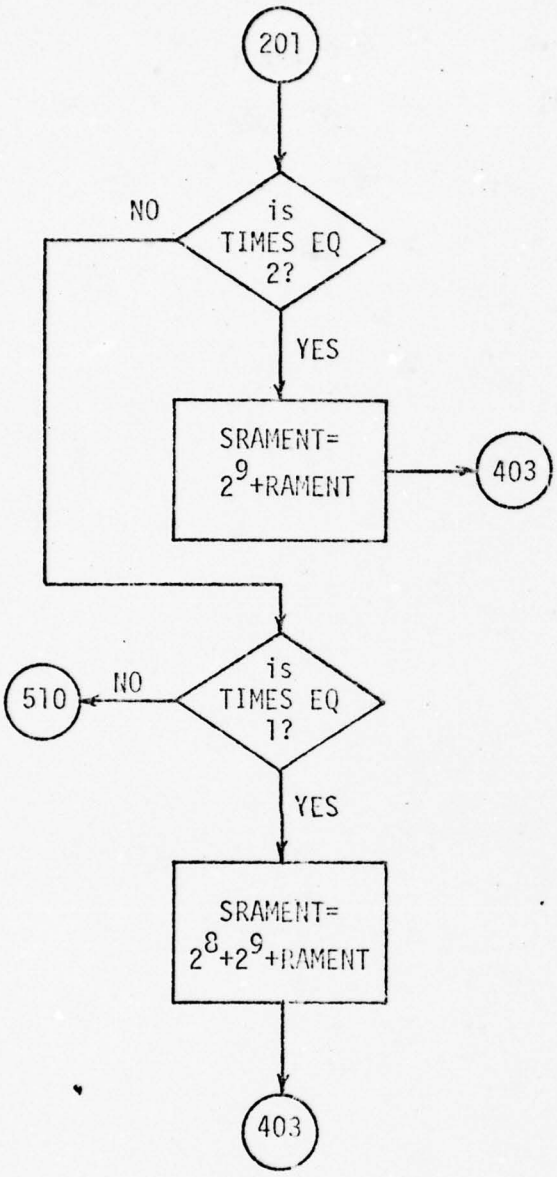
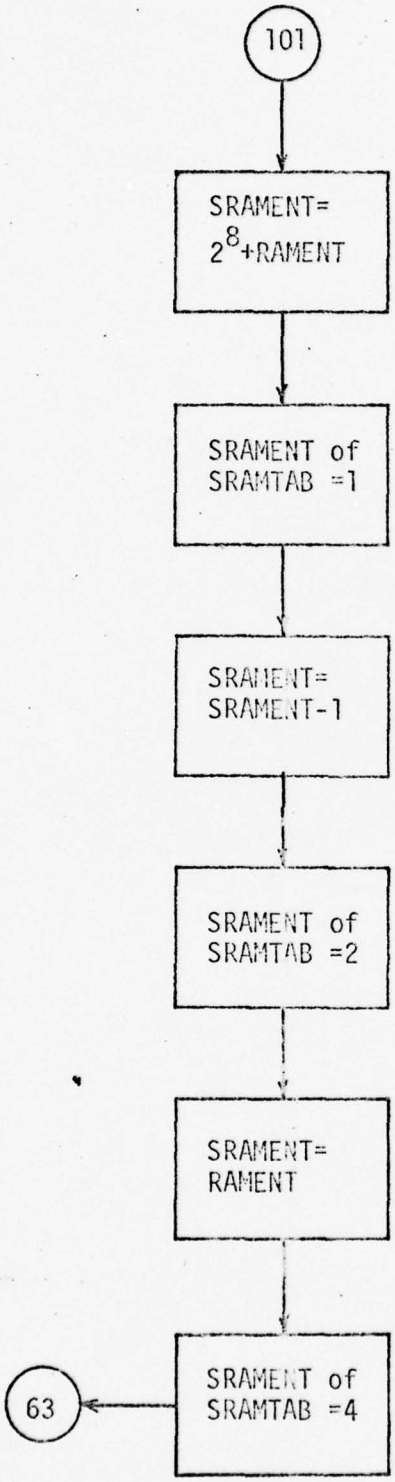


Figure 13 (cont).



The sector's width is 8 azimuth increments, therefore, one bit of the two frequency bits will be used for addressing and the other one will effectively be ignored by repeating the data destinations in addresses where its value is both one and zero.

Set the pointer to the address where frequency = 01.

Assign processor 1.

Decrement the pointer.

Assign processor 2.

Set the pointer to its normal value.

Assign processor 3.

Figure 13 (cont).

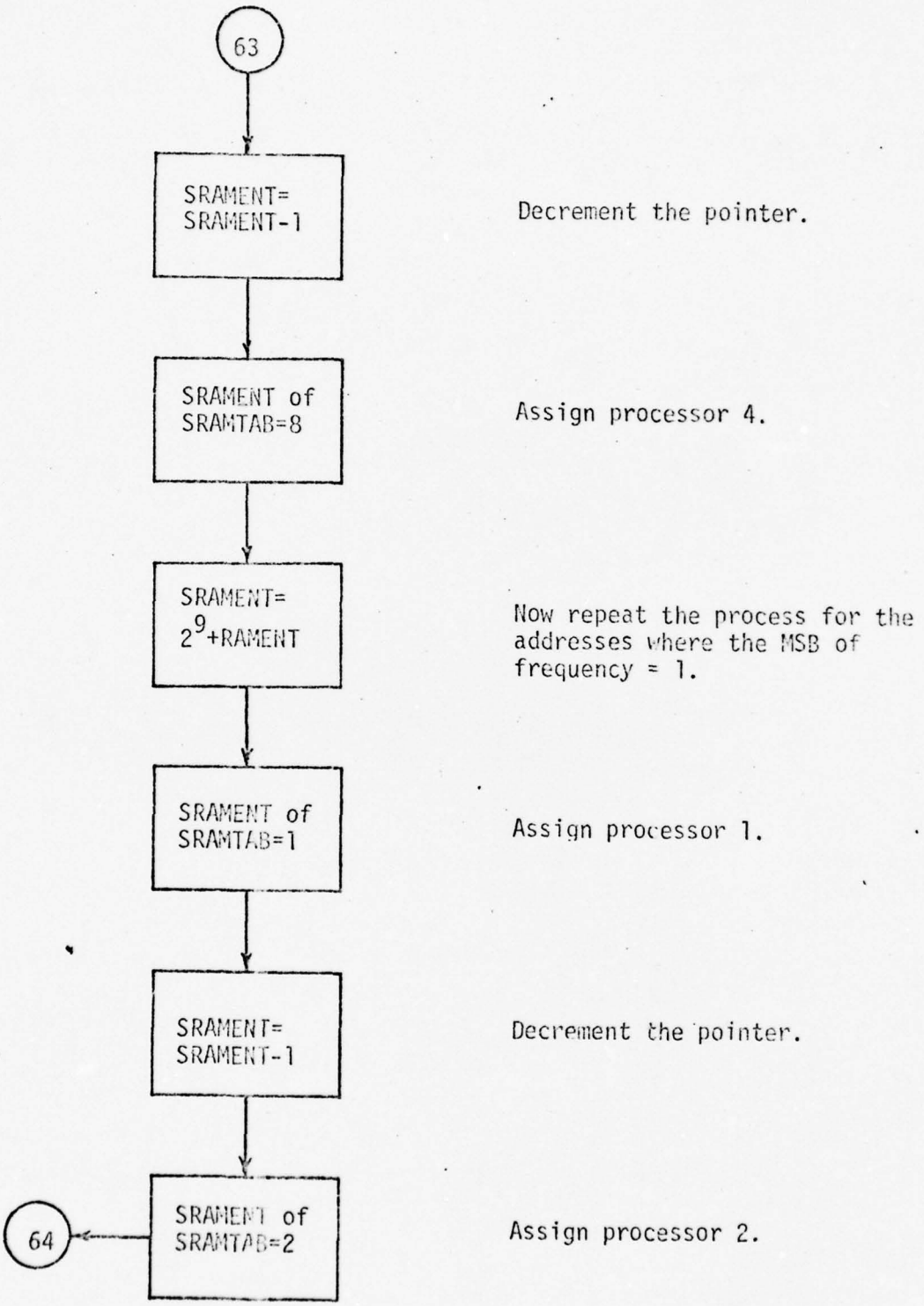
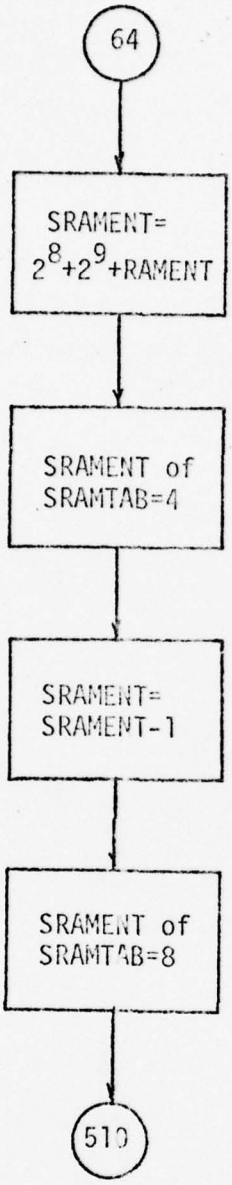


Figure 13 (cont).



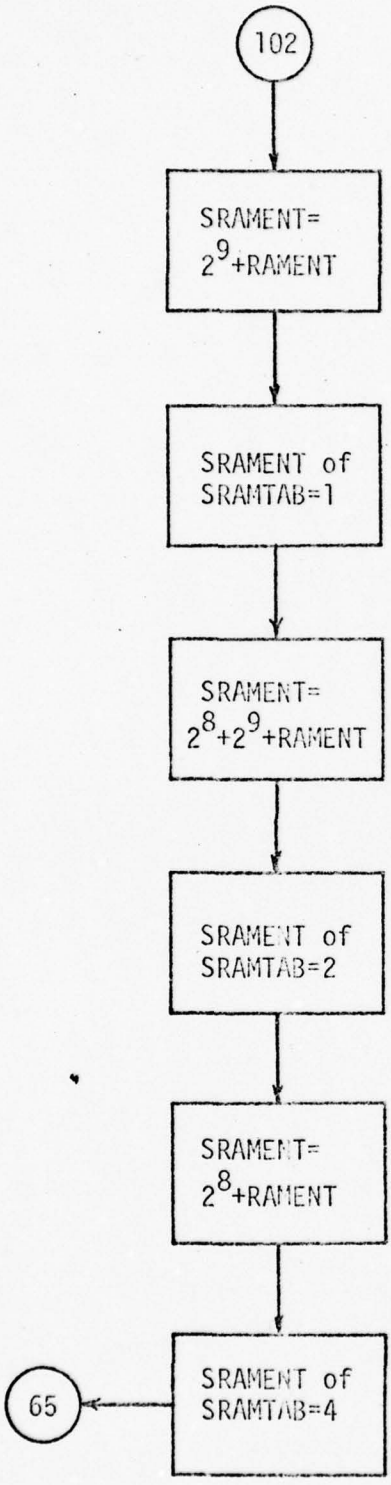
Set the pointer

Assign processor 3.

Decrement the pointer.

Assign processor 4.

Figure 13 (cont).



The sector's width is 4 azimuth increments, therefore, both bits of frequency will be used to help route the data, because the azimuth can no longer be divided.

Set the pointer to the address specified by frequency = 10 concatonated with AOA.

Assign processor 1.

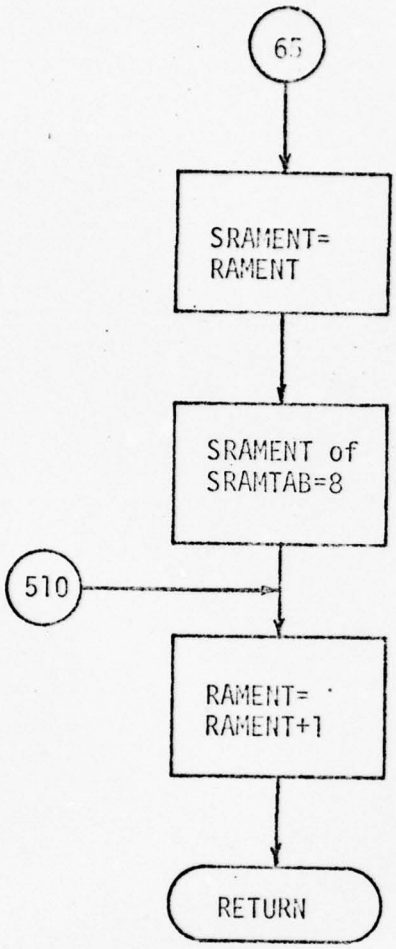
Set the pointer to the address specified by frequency = 11 concatonated with AOA.

Assign processor 2.

Set the pointer to the address specified by frequency = 01 concatonated with AOA.

Assign processor 3.

Figure 13 (cont).



Set the pointer to the address specified by frequency = 00 concatonated with AOA.

Assign processor 4.

Restore RAMENT to the value it had upon entry into this subroutine.

Figure 13 (cont).

Data Extraction Subroutine - DATAOUT

Functional Description. The subroutine DATAOUT is used to simulate the normal use of the logic arrays for data steering. It contains a table consisting of parameters from 16 fictitious emitters similar to those which would normally be received by the ECM system. These parameters are extracted from the input table and steered to their proper destinations via the logic arrays formatted by FSTRAM and SECRAM. For test purposes the subroutine prints out the sector and bin assignments on an output device.

Variables. The following variables are used, created, or modified by DATAOUT.

A. DATABUF is the input data table used for storing simulated radar parameters.

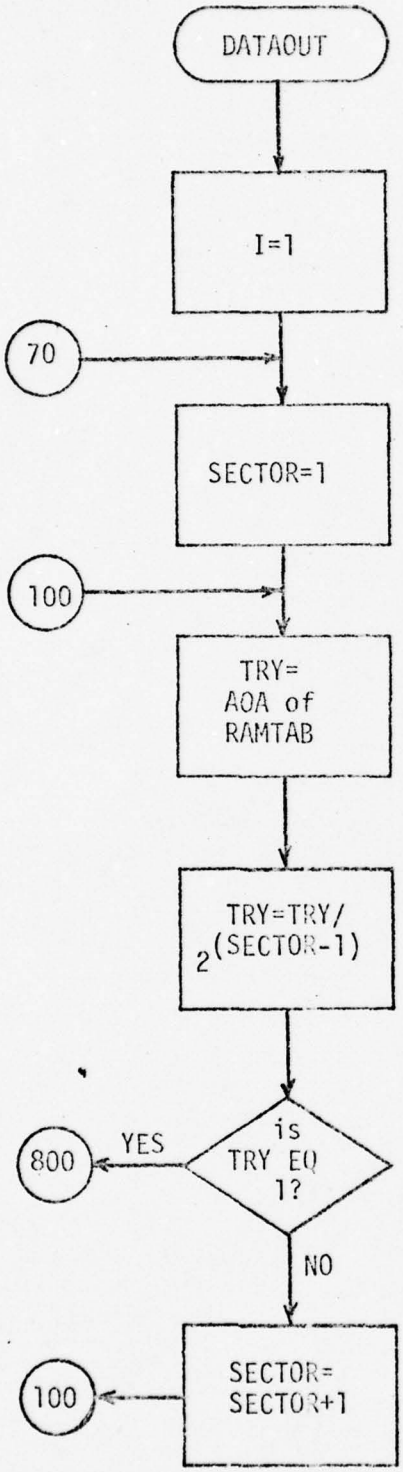
B. SECTOR is the general purpose table search pointer equal to the sector's numbers.

C. TRY is a local variable used to store the data destination (sector number) read from the first level logic array.

D. BIN is a variable indicating a microprocessor assignment.

E. STRY is a local variable used to store the data destination (bin assignment) read from the second level logic array.

Flowcharts. The detailed logic flow for subroutine DATAOUT is shown in Figure 14. A commented listing of the subroutine's implementation in FORTRAN IV is in Appendix A.



Set the counter I to 1 for extraction of the first emitter's parameters from DATABUF.

Initialize the sector search pointer.

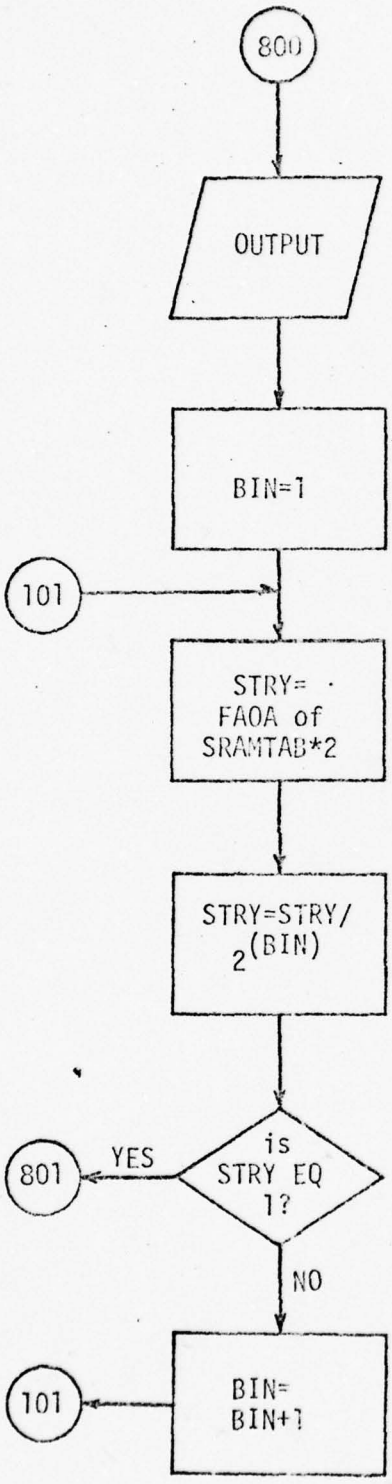
Extract the data destination from the first level logic array which is at the address specified by the emitter's AOA.

Translate the 16 bit quantity into a sector number.

Is this the proper sector for this data?

Increment SECTOR and try again.

Figure 14. Subroutine DATAOUT



Print out a message giving the chosen sector's number.

Initialize the bin search counter.

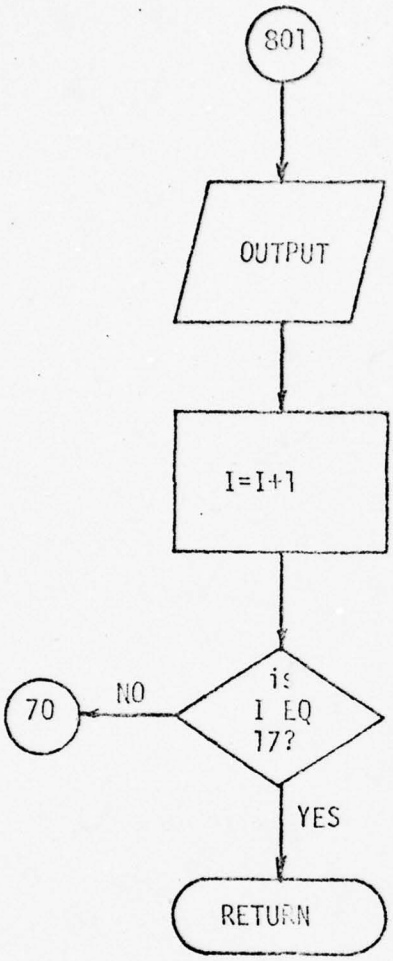
Extract the data destination from the second level logic array which is at the address specified by frequency concatonated with AOA.

Translate the 4 bit quantity into a bin number.

Is this the proper bin for this data?

Increment BIN and try again.

Figure 14 (cont).



Print out a message indicating the bin which was chosen.

Increment I to work on the next emitters' parameters.

Check to see if all of the emitter parameters have been extracted from DATABUF.

Figure 14 (cont).

VII Software Execution Time Analysis

To obtain an estimate of the time required to execute the microprocessor system's software programs, an assembly language equivalent of the routines must be derived. Since any microprocessor which is chosen for the actual system implementation will have its own unique machine and assembly language, the routines have been rewritten in a general assembly language and a rough order of magnitude estimate has been obtained. Two other assumptions are required. The first is an assumption of average conditions, i.e., subroutines are called and loops are executed a number of times commensurate with something less than the worst case. The other assumption which must be made is that of an average instruction time for an average microprocessor. The system requires a 16 bit microprocessor, therefore, an average instruction time of 5 microseconds would be representative of processors currently available [REF 11:56-186]. With data derived using the above conditions and assumptions a user can later derive best case or worst case estimates for his own set of conditions. The following tables contain the rough translation from FORTRAN IV to assembly language for each subroutine.

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time (usec)
SECTOR=0	MEMORY LOCATION =0	1	5
100: SECTOR=SECTOR+1	MOVE TO REGISTER	1	5
	INCREMENT REGISTER	16	80
103: IF(SECTOR.GT.16)GO TO 302	SUBTRACT IMMEDIATE	16	80
	DATA	16	80
	CONDITIONAL JUMP	8	40
IF(OVLTAB(SECTOR,1).NE.0) GO TO 102	MEMORY FETCH	16	80
	ADD	16	80
	DATA	16	80
	CONDITIONAL JUMP	8	40
GO TO 100	JUMP	16	80
102: OVLTAB(SECTOR,4)= OVLTAB(SECTOR,4)+1	MEMORY FETCH	6	30
	INCREMENT	6	30
IF(OVLTAB(SECTOR,4).LT.7) GO TO 100	SUBTRACT	6	30
	CONDITIONAL JUMP	6	30
300: OVLTAB(SECTOR,2)= OVLTAB(SECTOR,2)-1	MEMORY FETCH	3	15
	DECREMENT	3	15
311: OVLTAB(SECTOR,4)=0	MOVE IMMEDIATE TO MEMORY	3	15
	DATA	3	15
CUROVL=SECTOR	REGISTER TRANSFER	3	15

Table 1. Subroutine NORPROS

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
SECTOR=1	LOAD IMMEDIATE	3	15
	DATA	3	15
112: IF(OVLTAB(SECTOR,3).EQ.CUROVL) GO TO 104	MEMORY FETCH	16	80
	SUBTRACT REGISTER CONTENTS	16	80
	CONDITIONAL JUMP	8	40
SECTOR=SECTOR+1	INCREMENT REGISTER CONTENTS	15	75
GO TO 112	JUMP	15	75
104: EXPAND=SECTOR	REGISTER TRANSFER	8	40
OVLTAB(SECTOR,2)=1	MOVE IMMEDIATE TO MEMORY	8	40
	DATA	8	40
OVLTAB(SECTOR,3)=0	MOVE IMMEDIATE TO MEMORY	8	40
	DATA	8	40
GO TO 302	JUMP	8	40
113: SECTOR=CUROVL	REGISTER TRANSFER	8	40
IF(SECTOR.LE.15)GO TO 100	SUBTRACT IMMEDIATE	15	75
	DATA	15	75
	CONDITIONAL JUMP	8	40
GO TO 312	JUMP	1	5

Table 1 (cont)

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
302: IF(OVLTAB(CUROVL,2).LE.1) GO TO 500	MEMORY FETCH	4	20
	SUBTRACT	4	20
	DATA	4	20
	CONDITIONAL JUMP	2	10
GO TO 113	JUMP	2	10
500: OVLTAB(CUROVL,1)=0	MOVE IMMEDIATE TO MEMORY	2	10
	DATA	2	10
GO TO 113	JUMP	2	10
312: TIMES=TIMES-1	DECREMENT REGISTER	1*	5
IF(TIMES.EQ.0)GO TO 800	SUBTRACT IMMEDIATE	1*	5
	DATA	1*	5
	CONDITIONAL JUMP	1*	5
SECTOR=0	LOAD IMMEDIATE	1*	5
	DATA	1*	5
GO TO 100	JUMP	1*	5
	TOTALS	369	1845 usec.
	*Depends upon how many times NORPROS is executed before an interrupt.		

Table 1 (cont)

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
OVLTAB(OVLSECT)=1	LOAD IMMEDIATE	1	5
	DATA	1	5
CUROVL=OVLSECT	REGISTER TRANSFER	1	5
IF(OVLTAB(CUROVL,2).EQ.5)GO TO 777	MEMORY FETCH	1	5
	SUBTRACT IMMEDIATE	1	5
	DATA	1	5
	CONDITIONAL JUMP	1	5
IF(OVLTAB(CUROVL,2).EQ.4)GO TO 102	SUBTRACT IMMEDIATE	1	5
	DATA	1	5
	CONDITIONAL JUMP	1	5
GO TO 103	JUMP	1	5
102: OVLTAB(CUROVL,2)=5	MEMORY STORE	1	5
	DATA	1	5
EXPAND=CUROVL	REGISTER TRANSFER	1	5
SECTOR=CUROVL	REGISTER TRANSFER	1	5
GO TO 777	JUMP	1	5
103: OVLTAB(CUROVL,2)= OVLTAB(CUROVL,2)+1	REGISTER INCREMENT	1	5
	MEMORY STORE	1	5
SECTOR=1	LOAD IMMEDIATE	1	5
	DATA	1	5

Table 2. Subroutine INTSERV

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
105: IF(OVLTAB(SECTOR,1).NE.0) GO TO 104	MEMORY FETCH	8	40
	SUBTRACT IMMEDIATE	8	40
	DATA	8	40
	CONDITIONAL JUMP	4	20
IF(OVLTAB(SECTOR,2).NE.1)GO TO 104	SUBTRACT IMMEDIATE	6	30
	DATA	6	30
	CONDITIONAL JUMP	3	15
EXPAND=SECTOR	REGISTER TRANSFER	1	5
IF(OVLTAB(CUROVL,2).NE.1)GO TO 201	MEMORY FETCH	4	20
	SUBTRACT IMMEDIATE	4	20
	DATA	4	20
	CONDITIONAL JUMP	2	10
OVLTAB(SECTOR,3)=OVLTAB(CUROVL,3)	MEMORY FETCH	1	5
	MEMORY STORE	1	5
OVLTAB(CUROVL,3)=0	MEMORY STORE	1	5
	DATA	1	5
OVLTAB(SECTOR,2)=0	MEMORY STORE	1	5
	DATA	1	5
GO TO 106	JUMP	1	5
201: OVLTAB(SECTOR,3)=CUROVL	MEMORY STORE	1	5

Table 2 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
OVLTAB(SECTOR,2)=0	MEMORY STORE	1	5
	DATA	1	5
GO TO 106	JUMP	1	5
104: IF(SECTOR.GE.16)GO TO 700	SUBTRACT IMMEDIATE	8	40
	DATA	8	40
	CONDITIONAL JUMP	4	20
SECTOR=SECTOR+1	REGISTER INCREMENT	8	40
GO TO 105	JUMP	8	40
106: SECTOR=CUROVL	REGISTER TRANSFER	1	5
RETURN	RETURN	1	5
	TOTALS	135	675

Table 2 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
IF(EXPAND.GT.CUROVL)GO TO 201	REGISTER SUBTRACTION CONDITIONAL JUMP	1 1	5 5
FSTSECT=EXPAND	REGISTER TRANSFER	1	5
LSTSECT=CUROVL	REGISTER TRANSFER	1	5
GO TO 102	JUMP	1	5
201: FSTSECT=CUROVL	REGISTER TRANSFER	1	5
LSTSECT=EXPAND	REGISTER TRANSFER	1	5
102: SECTOR=1	LOAD IMMIDIATE DATA	1 1	5 5
RAMENT=1	LOAD IMMEDIATE DATA	1 1	5 5
208: IF(SECTOR.NE.FSTSECT) GO TO 202	REGISTER SUBTRACTION CONDITIONAL JUMP	8 4	40 20
GO TO 303	JUMP	4	20
202: IF(OVLTAB(SECTOR,2).NE.0) GO TO 203	MEMORY FETCH SUBTRACT IMMEDIATE DATA CONDITIONAL JUMP	8 8 8 4	40 40 40 20

TABLE 3. Subroutine FSTRAM

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
RAMENT=RAMENT+20	ADD IMMEDIATE	2	10
	DATA	2	10
GO TO 207	JUMP	2	10
203: IF(OVLTAB(SECTOR,2).NE.1) GO TO 204	MEMORY FETCH	6	30
	SUBTRACT IMMEDIATE	6	30
	DATA	6	30
	CONDITIONAL JUMP	3	15
RAMENT=RAMENT+16	ADD IMMEDIATE	2	10
	DATA	2	10
GO TO 207	JUMP	2	10
204: IF(OVLTAB(SECTOR,2).NE.2) GO TO 205	MEMORY FETCH	4	20
	SUBTRACT IMMEDIATE	4	20
	DATA	4	20
	CONDITIONAL JUMP	2	10
RAMENT=RAMENT+12	ADD IMMEDIATE	2	10
	DATA	2	10
GO TO 207	JUMP	2	10
205: IF(OVLTAB(SECTOR,2).NE.3) GO TO 206	MEMORY FETCH	3	15
	SUBTRACT IMMEDIATE	3	15
	DATA	3	15
	CONDITIONAL JUMP	2	10

Table 3 (cont)

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
RAMENT=RAMENT+8	ADD IMMEDIATE	2	10
	DATA	2	10
GO TO 207	JUMP	2	10
206: RAMENT=RAMENT+4	ADD IMMEDIATE	1	5
	DATA	1	5
207: SECTOR=SECTOR+1	REGISTER INCREMENT	7	35
IF(SECTOR.LE.16)GO TO 208	SUBTRACT IMMEDIATE	1	5
	DATA	1	5
	CONDITIONAL JUMP	1	5
303: IF(OVLTAB(SECTOR,2).NE.0) GO TO 210	MEMORY FETCH	8	40
	SUBTRACT IMMEDIATE	8	40
	DATA	8	40
	CONDITIONAL JUMP	4	20
WIDTH=20	LOAD IMMEDIATE	4	20
	DATA	4	20
GO TO 214	JUMP	4	20
210: IF(OVLTAB(SECTOR,2).NE.1) GO TO 211	MEMORY FETCH	6	30
	SUBTRACT IMMEDIATE	6	30
	DATA	6	30
	CONDITIONAL JUMP	3	15

Table 3 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
WIDTH=16	LOAD IMMEDIATE	3	15
	DATA	3	15
GO TO 214	JUMP	3	15
211: IF(OVLTAB(SECTOR,2).NE.2) GO TO 212	MEMORY FETCH	4	20
	SUBTRACT IMMEDIATE	4	20
	DATA	4	20
	CONDITIONAL JUMP	2	10
WIDTH=12	LOAD IMMEDIATE	2	10
	DATA	2	10
GO TO 214	JUMP	2	10
212: IF(OVLTAB(SECTOR,2).NE.3) GO TO 213	MEMORY FETCH	2	10
	SUBTRACT IMMEDIATE	2	10
	DATA	2	10
	CONDITIONAL JUMP	1	5
WIDTH=8	LOAD IMMEDIATE	1	5
	DATA	1	5
GO TO 214	JUMP	1	5
WIDTH=4	LOAD IMMEDIATE	1	5
	DATA	1	5
214: LSTENT=RAMENT+WIDTH	REGISTER ADD	8	40
	REGISTER TRANSFER	8	40

Table 3 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
105: RAMCONT=2**(SECTOR-1)	SUBTRACT IMMEDIATE	8	40
	DATA	8	40
	MULTIPLY	8	320*
	DATA	8	40
RAMTAB(RAMENT)=RAMCONT	MEMORY STORE	72	360
104: RAMENT=RAMENT+1	REGISTER INCREMENT	72	360
IF(RAMENT.NE.LSTENT)GO TO 105	REGISTER SUBTRACT	72	360
	CONDITIONAL JUMP	36	180
CALL SECRAM	SUBROUTINE CALL	8	40
IF(SECTOR.EQ.LSTSECT)GO TO 500	REGISTER SUBTRACT	7	35
	CONDITIONAL JUMP	4	20
SECTOR=SECTOR+1	REGISTER INCREMENT	7	35
GO TO 303	JUMP	7	35
RETURN	RETURN	1	5
	TOTALS	622	3110
	*assumes a 40 usec. multiply time		

Table 3 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
RAMENT=RAMENT-1	REGISTER DECREMENT	1	5
SRAMENT=RAMENT	REGISTER TRANSFER	1	5
BWIDTH=WIDTH/4	SHIFT	1	5
	SHIFT	1	5
IF(BWIDTH.EQ.2)GO TO 101	SUBTRACT IMMEDIATE	1	5
	DATA	1	5
	CONDITIONAL JUMP	1	5
IF(BWIDTH.EQ.1)GO TO 102	SUBTRACT IMMEDIATE	1	5
	DATA	1	5
	CONDITIONAL JUMP	1	5
TIMES=4	LOAD IMMEDIATE	1	5
	DATA	1	5
403: BIN=3	LOAD IMMEDIATE	1	5
	DATA	1	5
104: BINCONT=2**BIN	LOAD IMMEDIATE	4	20
	DATA	4	20
	DECREMENT BIN	4	20
	CONDITIONAL JUMP	4	20
	SHIFT	4	20
WIDCNT=SRAMENT-BWIDTH	REGISTER SUBTRACT	4	20
	REGISTER TRANSFER	4	20

Table 4. Subroutine SECRAM

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
105: SRAMTAB(SRAMENT)=BINCONT	MEMORY STORE	4	20
SRAMENT=SRAMENT-1	REGESTER DECREMENT	3	15
IF(SRAMENT.NE.WIDCNT)GO TO 105	SUBTRACT IMMEDIATE	3	15
	DATA	3	15
	CONDITIONAL JUMP	2	10
IF(BIN.EQ.0)GO TO 106	SUBTRACT IMMEDIATE	3	15
	DATA	3	15
	CONDITIONAL JUMP	2	10
BIN=BIN-1	REGISTER DECREMENT	3	15
*****Allow for the repetition of the above 5 FORTRAN statements an average of 3 times per execution.		52	260
BIN=BIN-1	REGISTER DECREMENT	3	15
GO TO 104	JUMP	3	15
106: TIMES=TIMES-1	REGISTER DECREMENT	3	15
IF(TIMES.EQ.3)GO TO 200	SUBTRACT IMMEDIATE	3	15
	DATA	3	15
	CONDITIONAL JUMP	2	10
GO TO 201	JUMP	1	10
*****Allow for the repetition of the previous 13 FORTRAN statements 4 times per execution.		232	1160

Table 4 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
200: SRAMENT=(2**8)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
GO TO 403	JUMP	1	5
201: IF(TIMES.EQ.2)GO TO 202	SUBTRACT IMMEDIATE	3	15
	DATA	3	15
	CONDITIONAL JUMP	2	10
GO TO 203	JUMP	2	10
202: SRAMENT=(2**9)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
GO TO 403	JUMP	1	5
203: IF(TIMES.EQ.1)GO TO 204	SUBTRACT IMMEDIATE	2	10
	DATA	2	10
	CONDITIONAL JUMP	1	5
GO TO 510	JUMP	1	5
204: SRAMENT=(2**8)+(2**9)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
GO TO 403	JUMP	1	5

Table 4 (cont).

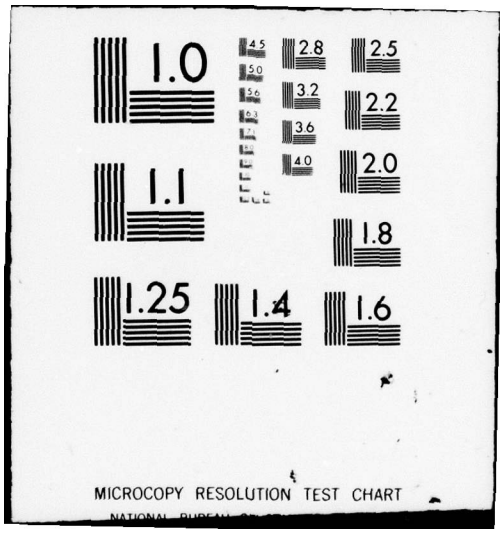
FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
101: SRAMENT=(2**8)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=1	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=SRAMENT-1	REGISTER DECREMENT	1	5
SRAMTAB(SRAMENT)=2	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=RAMENT	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=4	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=SRAMENT-1	REGISTER DECREMENT	1	5
SRAMTAB(SRAMENT)=8	MEMORY STORE	1	5
	DATA	1	5
107: SRAMENT=(2**9)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=1	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=SRAMENT-1	REGISTER DECREMENT	1	5

Table 4 (cont).

AD-A064 677 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB, OHIO F/G 17/4
A PARALLEL MICROPORCESSOR ARCHITECTURE FOR ELECTRONIC COUNTERMEASURES PROCESSING. DEC 78 HENRY, KENNETH L. MASTERS THESIS
UNCLASSIFIED REPT. NO. AFIT/GE/EE/78-26 NL

2 OF 2
AD A 064677

END
DATE FILMED
11-81
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
SRAMTAB(SRAMENT)=2	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=(2**8)+(2**9)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=4	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=SRAMENT-1	REGISTER DECREMENT	1	5
SRAMTAB(SRAMENT)=8	MEMORY STORE	1	5
	DATA	1	5
GO TO 510	JUMP	1	5
102: SRAMENT=(2**8)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=1	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=(2**9)+(2**8)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=2	MEMORY STORE	1	5
	DATA	1	5

Table 4 (cont).

FORTRAN IV	ASSEMBLY LANGUAGE	# of Times Executed	Total Time(usec)
SRAMENT=(2**9)+RAMENT	ADD IMMEDIATE	1	5
	DATA	1	5
	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=4	MEMORY STORE	1	5
	DATA	1	5
SRAMENT=RAMENT	REGISTER TRANSFER	1	5
SRAMTAB(SRAMENT)=8	MEMORY STORE	1	5
	DATA	1	5
RAMENT=RAMENT+1	REGISTER INCREMENT	1	5
RETURN	RETURN	1	5
	TOTALS	449	2245

Table 4 (cont).

Analysis

The master processor's executive subroutine, NORPROS, requires an average of approximately 2 milliseconds per execution. This estimate can fluctuate depending upon the number of existing overloaded sectors before NORPROS' execution and the number of new overloads which occur during and after its execution. In a benign environment there are few overloads to reduce and NORPROS can execute in approximately 1 millisecond. In a very dense environment, i.e., all sectors are either overloaded or expanded, NORPROS will either not be allowed to execute because of constant interrupts or will take more than 3 milliseconds per execution because of the large number of overload reductions which must be performed. The most important piece of data obtained from examination of NORPROS is an average execution time number which can be used to make an intelligent choice for the maximum number of NORPROS execution times a sector will remain overloaded before a reduction is attempted.

INTSERV is a short, fast subroutine which determines an overloaded sector's number, finds an expandable sector, and passes those data to FSTRAM. Its average execution time is approximately .7 milliseconds and is not very dependent upon external conditions.

The subroutines which reformat the logic arrays, FSTRAM and SECRAM, are heavily dependent upon the system's status, the density of the external pulse environment, and the locations of the sectors which overload. As the difference between the overloaded sector's number and the expandable sector's number increases, the execution time for both FSTRAM and SECRAM increases due to the larger portions of logic arrays which must be reformatted. On the average, FSTRAM will take over 3 milliseconds to execute and SECRAM will take 2.3 milliseconds. The essential assumption which was made in deriving

these numbers was that the overloaded sector and the expanded sector were 8 sectors apart.

Summary

Using the numbers derived from the preceeding analysis, the time required to restructure the microprocessor network after a sector overloads can be estimated. First, an interrupt occurs and .7 milliseconds is spent in INTSERV. INTSERV then calls FSTRAM which reformats one sector's entries in the first level logic array. After finishing its modification on one sector, FSTRAM calls SECRAM to reformat that sector's second level logic array. SECRAM returns control to FSTRAM and the process is repeated until all logic array entries which must be reformatted are modified. The total elapsed time is approximately 6 milliseconds. Meanwhile, processing in the non-overloaded sectors, i.e., all but the one which interrupted NORPROS, continues normally. Processing does not cease in the overloaded sector, however. Pulses are still admitted into the sector's FIFO on a space available basis as the sector's microprocessors extract data for processing. An extensive analysis of expected pulse density for representative scenarios would be required before a quantitative estimate of how many pulses would be lost during the 6 millisecond delay could be obtained.

VIII Conclusions and Recommendations

General Observations

Electronic countermeasures systems present a data processing problem which is definetely amenable to solution via application of a parallel microprocessor architecture. The input data rate is high and unpredictable making a single instruction stream processor unfeasible. There are, however, parameters in the data stream which can be used to vector portions of the data to separate processors. The multiple instruction stream network which was investigated provides sufficient division of the processing task to allow parallel data processing without the addition of overly cumbersome control hardware and software. A master processor, executive control software, programmed logic arrays, and First-In-First-Out memories provide the necessary monitoring and control to allow 64 independent microprocessors to operate in parallel. The architecture is flexible and adaptable, and reconfigures itself to cope with the unpredictable nature of electronic countermeasures data.

Overall Results

The system software and hardware design was simulated and exercised against a test scenerio and fictitious radar data. Test results, presented in Appendix C, highlight the microprocessor's response to system overloads. The master processor's overload handling software allowed for the orderly reconfiguration of the network as overloads occurred and the restoration of the network as overloads diminished. Additional test results exhibit

the capability of the programmed logic arrays to steer the incoming radar pulse descriptor words to their proper processing locations while adding very little time delay between receipt of a pulse and the start of processing.

Software Subroutine Improvements

In general, the subroutines are constrained to a specific set of initial conditions. Changing the test scenerio or input data stream requires a software modification. To prevent possible configuration control problems and to provide for easier testing, the subroutines should be modified to accept user specified initial conditions and should be made interactive. The test scenerio can be generated in advance and loaded via punched cards, paper tape, or magnetic tape. The specific radar parameters of the threat emitters can be loaded at run time. Initial sector overload status can be specified by the user during the actual test.

Another software improvement which can be made involves the general nature of the architecture itself. For test purposes, the architecture presented in this thesis was assumed to consist of sixteen sectors and four microprocessors per sector. For future investigations a modification can be made to allow various different configurations by allowing changes in the number of sectors, sector spacing, and microprocessor assignments to the sectors. Tradeoff studies can be made to analyze the impact of different architecture configurations on system overloads and throughput.

Potential Hardware Modifications

One of the most difficult decisions to make when partitioning the azimuth coverage of the network into sectors is on sector size versus the total number of sectors. Because the actual environment is unknown a priori

the architecture must be general in nature and must be flexible, resulting in a finite amount of control hardware and software. If, however, the system could somehow measure its environment and adapt itself, the amount of control hardware and software could be reduced and the system's performance increased. One way to provide the required environmental data is to place hardware pulse counters at each sector FIFO (counting the FIFO enable signals) to keep track of the quantity of incoming radar pulse descriptor words. This data can be monitored by the master processor which can then adapt the network's configuration.

One other improvement which could be made is to employ one FIFO for each microprocessor instead of one FIFO per sector. This tradeoff is discussed in Chapter V and would result in a four fold increase in each sector's throughput capability as well as the elimination of the FIFO control circuitry.

Recommendations for Further Study

As it is presented in this thesis, the microprocessor network requires a set of initial conditions and a fixed scenario to be tested and evaluated. Dynamic stimulation of the processor architecture can be achieved by modeling a typical electronic countermeasures environment to create input data for the network. Implemented in a minicomputer, this stimulation can be accomplished in real time and the parallel microprocessor concept can be extensively tested.

Another area for further study is the actual fabrication of portions of the system's hardware. Two candidates for fabrication are the data steering logic and the master processor. Logic array programs can be created, stored in actual RAM's, and tested to provide further insight into

system time delays. Also, the master processor software can be programmed for and executed in an actual microprocessor to test its performance.

As an alternative to an extremely flexible architecture configuration, made necessary by the unpredictable data environment, an extensive analysis can be accomplished to determine an optimum parallel architecture. This analysis would require the examination of actual weapon system scenarios and would necessarily be classified.

One final area for potential future study is the complexity of the control overhead inherent in a parallel processing complex. The overload handling software presented in this thesis is cumbersome and could probably be streamlined to operate more efficiently. The number of intermediate states between situation normal and a severe overload should be examined as well as the transitions between these states.

Bibliography

1. Electronic Countermeasures, Prepared by the Institute of Science and Technology of the University of Michigan, 1965.

2. Introduction to Radar Systems, Merrill I. Skolnik, New York, 1962.

3. F-15 Tactical Electronic Warfare System (TEWS) Specification.

4. AN/ALQ-131 Electronic Countermeasures Pod System Specification.

5. Flynn, Michael J , "Very High-Speed Computing Systems," Proceedings of the IEE, December, 1966, Pages 1901-1909.

6. Stone, Harold S., Editor, Introduction to Computer Architecture, Science Research Associate, Inc., 1975.

7. Hobbs, L.C., et. al., Editors, Parallel Processor Systems, Technologies, and Applications, Spartan Books, New York, 1970.

8. Enslow, Philip H., Jr., Editor, Multiprocessors and Parallel Processing, John Wiley and Sons, New York, 1974.

9. B-1 Radio Frequency Surveillance/Electronic Countermeasures Systems Specification No. CP07878139A0600D, 23 July 1976.

10. AFSC Handbook DH 2-7 System Survivability (U), Second Edition, Revision No. 5, 20 November 1974.

11. "Microprocessor Survey," Electronic Design, 11 October 1977, Pages 56-185.

Appendix A

```
C*****************************************************************
C*****************************************************************
C*************************PROGRAM SENERIO*************************
C*****************************************************************
C*****************************************************************
C*****THE PURPOSE OF SENERIO IS TO TEST THE OVERLOAD HANDLING
C**CAPABILITIES OF THE MICROPROCESSOR ARCHITECTURE.  SUBROUTINE
C**INTSERV HANDLES THE CREATION OF OVERLOADS WHILE SUBROUTINE
C**NORPROS ATTEMPTS TO REDUCE OVERLOADED CONDITIONS TO NORMAL.
C
C
C
C
      PROGRAM SENERIO(INPUT,OUTPUT,TAPE6=OUTPUT)
      IMPLICIT INTEGER(A-Z)
      DIMENSION OVLTAB(16,4)
C
C
C*****ARRAY "OVLTAB" IS THE SYSTEM'S OVERLOAD TABLE. THE FOUR
C**VARIABLES PER ENTRY ARE DEFINED AS FOLLOWS:  OVLTAB(N,1) IS
C**THE OVERLOAD FLAG FOR SECTOR N, OVLTAB(N,2) IS OVLSTAT WHICH
C**INDICATES HOW SEVERELY SECTOR N IS OVERLOADED, OVLTAB(N,3) IS
C**A POINTER WHICH POINTS TO THE SECTOR WHICH CAUSED SECTOR N TO
C**BE EXPANDED, OVLTAB(N,4)IS VARIABLE OREDCNT WHICH IS A COUNT
C**TO KEEP TRACK OF WHEN TO REDUCE SECTOR N'S OVERLOAD STATUS.
C
C
C*****THE FOLLOWING STATEMENTS INITIALIZE THE SYSTEM'S OVERLOAD
C**TABLE.  INITIALLY, SECTORS 7 & 10 ARE IN OVERLOAD STATE 2 AND
C**SECTORS 8 & 9 ARE IN OVERLOAD STATE 3.  SECTOR 1 HAS BEEN
C**EXPANDED BY SECTOR 8, SECTOR 2 HAS BEEN EXPANDED BY SECTOR 9,
C**SECTOR 3 HAS BEEN EXPANDED BY SECTOR 8, SECTOR 4 HAS BEEN
C**EXPANDED BY,SECTOR 9, SECTOR 5 HAS BEEN EXPANDED BY SECTOR 7,
C**AND SECTOR 6, HAS BEEN EXPANDED BY SECTOR 10.  THE OVERLOAD
C**REDUCTION COUNTS FOR SECTORS 7-10 ARE 0,2,3,0 RESPECTIVELY.
C
C
      DO 100 I=1,6
100   OVLTAB(I,1)=0
      DO 101 I=7,10
101   OVLTAB(I,1)=1
      DO 102 I=11,16
102   OVLTAB(I,1)=0
      DO 103 I=1,6
103   OVLTAB(I,2)=0
      DO 104 I=11,16
104   OVLTAB(I,2)=1
      DO 105 I=7,16
105   OVLTAB(I,3)=0
      DO 106 I=1,7
```

```
106     OVLTAB(I,4)=0
        DO 107 I=10,16
107     OVLTAB(I,4)=0
        OVLTAB(7,2)=2
        OVLTAB(8,2)=3
        OVLTAB(9,2)=3
        OVLTAB(10,2)=2
        OVLTAB(1,3)=8
        OVLTAB(2,3)=9
        OVLTAB(3,3)=8
        OVLTAB(4,3)=9
        OVLTAB(5,3)=7
        OVLTAB(6,3)=10
        OVLTAB(8,4)=2
        OVLTAB(9,4)=3
C
C*****PRINT OUT THE INITIAL OVERLOAD TABLE
C
        WRITE(6,900)
900     FORMAT(2(/),8X,"INITIAL OVERLOAD TABLE",2(/))
        WRITE(6,901)((OVLTAB(M,N),N=1,4),M=1,16)
901     FORMAT((5X,4(3X,I3)))
C
C*****THE SCENERIO FOLLOWS.  THE VARIABLE TIMES SIMULATES HOW
C**MANY TIMES THE NORMAL PROGRAM WHICH HAS OVERLOAD REDUCTION AS
C**ONE OF ITS TASKS, WOULD BE RUN.  THE VARIABLE IS SET AND THE
C**SUBROUTIN NORPROS IS CALLED.  THE VARIABLE OVLSECT SIMULATES
C**DATA WHICH WOULD NORMALLY BE RETURNED FROM AN INTERRUPT
C**HANDLING SUBROUTINE IF THE SIXTEEN SECTORS SIGNALLED OVERLOADS
C**VIA PRIORITIZED INTERRUPTS.  AFTER OVLSECT IS SET, THE
C**SUBROUTINE INTSERV IS CALLED. THIS SCENERIO SIMULATES A FLIGHT
C**PATH TOWARDS AN DENSE ENVIRONMENT WHERE THE SECTORS ON EITHER
C**SIDE OF 0 DEGREES AZIMUTH OVERLOAD AND THEN THE FLIGHT PATH
C**CHANGES WITH A TURN TO THE RIGHT CAUSING THE SECTORS ON THE
C**LEFT SIDE TO OVERLOAD.
C
C
C*****THE REAR SECTORS ARE OVERLOADED BUT NO INTERRUPTS ARE
C*PENDING SO NORPROS WILL NOW HAVE A CHANCE TO REDUCE THE
C*OVERLOAD STATUS OF SOME SECTORS.  IF THIS IS DONE TOO SOON,
C*I.E.,THE SECTORS ARE STILL OVERLOADED THEY WILL RE-OVERLOAD.
C
C
        TIMES=8
        CALL NORPROS(TIMES,OVLTAB)
C
C
C*****THE AIRCRAFT IS NOW FLYING TOWARDS A HIGH DENSITY RADIO
C*FREQUENCY ENVIRONMENT AND  THE SECTORS ON EITHER SIDE OF THE
C*NOSE WILL BEGIN OVERLOADING.
```

```
C
C*****SECTOR ONE OVERLOADS
C
      OVLSECT=1
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****OVERLOAD REDUCTION ATTEMPT FOLLOWS
C
      TIMES=10
      CALL NORPROS(TIMES,OVLTAB)
C
C*****SECTOR ONE RE-OVERLOADS
C
      OVLSECT=1
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****SECTOR SIXTEEN OVERLOADS
C
      OVLSECT=16
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****OVERLOAD REDUCTION ATTEMPT FOLLOWS
C
      TIMES=5
      CALL NORPROS(TIMES,OVLTAB)
C
C*****SECTOR ONE RE-OVERLOADS
C
      OVLSECT=1
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****SECTOR SIXTEEN RE-OVERLOADS AND SECTORS FIFTEEN AND TWO
C*OVERLOAD FOR THE FIRST TIME.
C
      OVLSECT=16
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=15
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=2
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****NORPROS IS ALLOWED TO EXECUTE ONLY THREE TIMES BECAUSE OF
C*THE DENSE ENVIRONMENT.  THIS TIME IS ARTIFICIALLY SHORT TO ALLOW
C*FOR MORE TESTING OF THE SYSTEM'S OVERLOAD HANDLING CAPABILITY.
C
      TIMES=3
      CALL NORPROS(TIMES,OVLTAB)
C
C*****SECTORS ONE, TWO, FIFTEEN, AND SIXTEEN RE-OVERLOAD.
```

```
      OVLSECT=1
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=16
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=15
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=2
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****OVERLOAD REDUCTION ATTEMPT FOLLOWS
C
      TIMES=2
      CALL NORPROS(TIMES,OVLTAB)
C
C*****AS THE AIRCRAFT GETS CLOSER TO THE SIMULATED DENSE
C*ENVIRONMENT, THE OVERLOADS IN THE NOSE SECTORS GET MORE
C*SEVERE.
C
      OVLSECT=1
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=16
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=15
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=2
      CALL INTSERV(OVLSECT,OVLTAB)
C
C*****THE AIRCRAFT NOW BEGINS A TURN TO THE RIGHT TO PREVENT
C*OVERFLIGHT OF A HIGHLY DEFENDED TARGET. AS A RESULT,THE SECTORS
C*ON THE LEFT SIDE OF THE AIRCRAFT BEGIN TO OVERLOAD AND THE NOSE
C*SECTORS BEGIN TO RETURN TO NORMAL.
C
      TIMES=1
      CALL NORPROS(TIMES,OVLTAB)
      OVLSECT=9
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=10
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=12
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=11
      CALL INTSERV(OVLSECT,OVLTAB)
      TIMES=1
      CALL NORPROS(TIMES,OVLTAB)
      OVLSECT=11
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=9
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=12
```

```
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=10
      CALL INTSERV(OVLSECT,OVLTAB)
      TIMES=2
      CALL NORPROS(TIMES,OVLTAB)
      OVLSECT=10
      CALL INTSERV(OVLSECT,OVLTAB)
      OVLSECT=11
      CALL INTSERV(OVLSECT,OVLTAB)
      TIMES=5
      CALL NORPROS(TIMES,OVLTAB)
      TIMES=10
      CALL NORPROS(TIMES,OVLTAB)
      TIMES=35
      CALL NORPROS(TIMES,OVLTAB)
      STOP
      END
```

```
C
C
C*****SUBROUTINE NORPROS IS A SORT OF OVERALL EXECUTIVE FOR THE
C**PROCESSOR SYSTEM.  NORMALLY THE SUBROUTINE WOULD BE CONSTANTLY
C**EXECUTING UNTIL AN OVERLOADED SECTOR CAUSED AN INTERRUPT.
C**FOR TEST PURPOSES THE INTERRUPTS ARE SIMULATED AND NORPROS IS
C**ONLY EXECUTED THE NUMBER OF TIMES SPECIFIED BY THE VARIABLE
C**TIMES.  ALSO, FOR TEST PURPOSES, THE AMOUNT OF TIME SPENT
C**EXECUTING NORPROS IS ARTIFICIALLY SHORT TO ALLOW MORE TESTING
C**OF THE SYSTEM'S OVERLOAD HANDLING CAPABILITY.
C
C
C
      SUBROUTINE NORPROS(TIMES,OVLTAB)
      IMPLICIT INTEGER(A-Z)
      DIMENSION OVLTAB(16,4)
C
      WRITE(6,950)
950   FORMAT(2(/),5X,"NORPROS WILL NOW TRY TO REDUCE OVERLOADS")
      WRITE(6,951)TIMES
951   FORMAT(1(/),5X,"NORPROS WILL BE EXECUTED",1X,I2,1X,"TIMES")
C
      SECTOR=0
C*****INITIALIZE THE SECTOR SEARCH COUNTER
100   SECTOR=SECTOR+1
C*****INCREMENT THE SECTOR SEARCH COUNTER
103   IF(SECTOR.GT.16)GO TO 312
C*****TEST SECTOR, IF IT IS GREATER THAN 16 ALL SECTORS HAVE BEEN
C*****CHECKED
      IF(OVLTAB(SECTOR,1).NE.0)GO TO 102
C*****TEST THE OVERLOAD FLAG FOR THIS SECTOR.  IF IT IS ZERO THAN
C*THE SECTOR IS NOT OVERLOADED AND NOTHING NEEDS TO BE DONE-GO TO
C*THE NEXT SECTOR.  IF HOWEVER, THE OVERLOAD FLAG IS ONE THAN THE
C*SECTOR IS OVERLOADED AND THE OVERLOAD REDUCTION COUNT MUST BE
C*INCREMENTED-JUMP TO THAT SECTION OF CODE.
      GO TO 100
C*****JUMP BACK AND CHECK THE NEXT SECTOR
C
102   OVLTAB(SECTOR,4)=OVLTAB(SECTOR,4)+1
C*****INCREMENT THE OVERLOAD REDUCTION COUNT
      IF(OVLTAB(SECTOR,4) .LT. 7) GO TO 100
C*****IF THE OVERLOAD REDUCTION COUNT IS LESS THAN SEVEN GO CHECK
C*THE NEXT SECTOR, OTHERWISE, JUMP TO THE SECTION OF CODE WHICH
C*INSTITUTES A REDUCTION IN THIS SECTORS OVERLOAD STATUS.
C
300   OVLTAB(SECTOR,2)=OVLTAB(SECTOR,2)-1
C*****DECREMENT THE SECTOR'S OVERLOAD STATUS.
311   OVLTAB(SECTOR,4)=0
C*****RESET THE SECTOR'S OVERLOAD FLAG
```

```
      CUROVL=SECTOR
C*****SET THE VARIABLE CUROVL TO THE NUMBER OF THE SECTOR WHOSE
C*OVERLOAD STATUS IS TO BE REDUCED.  NORMALLY,CUROVL IS A
C*VARIABLE USED BY SUBROUTINE INTSERV TO INDICATE AN OVERLOADED
C*SECTOR WHOSE AZIMUTH COVERAGE IS TO BE SHRUNK BY SUBROUTINE
C*FSTRAM.  IN THIS CASE IT IS USED MERELY TO SAVE THE VALUE OF
C*SECTOR AND WILL BE RESTORED LATER.
C
      SECTOR=1
C*****INITIALIZE THE SECTOR SEARCH COUNTER
112   IF (OVLTAB(SECTOR,3).EQ.CUROVL) GO TO 104
C*****IF THE ABOVE TEST IS TRUE THEN THE EXPANDED SECTOR HAS BEEN
C*FOUND.
      SECTOR=SECTOR+1
      GO TO 112
C
104   EXPAND=SECTOR
C*****SET THE VARIABLE EXPAND TO THE SECTOR NUMBER OF THE SECTOR
C*WHICH WAS EXPANDED.  EXPAND IS USED BY SUBROUTINE FSTRAM TO
C*REFORMAT THE FIRST LEVEL RAM'S LOGIC ARRAY.
      OVLTAB(SECTOR,2)=1
C*****RESTORE THE EXPANDED SECTOR'S OVERLOAD STATUS TO NORMAL
      OVLTAB(SECTOR,3)=0
C*****RESET THE EXPANDED SECTOR'S EXPANDER POINTER.
      GO TO 302
C*****GO PERFORM A TEST TO SEE IF THIS SECTOR'S OVERLOAD STATUS
C*HAS BEEN REDUCED ALL THE WAY BACK TO NORMAL(OVLTAB(CUROVL,2)=1)
C*IF IT HAS THEN RESET THE OVERLOAD INDICATOR FLAG.
C
113   SECTOR=CUROVL
C*****RESTORE THE SECTOR POINTER TO THE CURRENTLY OVERLOADED
C*SECTOR
      IF(SECTOR.LE.15)GO TO 100
      GO TO 312
C
302   IF(OVLTAB(CUROVL,2).LE.1)GO TO 500
      GO TO 113
C
500   OVLTAB(CUROVL,1)=0
      GO TO 113
C
312   TIMES=TIMES-1
C*****DECREMENT THE COUNTER WHICH KEEPS TRACK OF THE NUMBER OF
C*TIMES THIS SUBROUTINE MUST BE EXECUTED.  IF IT HAS REACHED
C*ZERO THEN PRINT OUT THE CURRENT OVERLOAD TABLE AND EXIT BACK
C*TO THE MAIN PROGRAM, OTHERWISE GO BACK TO THE BEGINNING OF
C*THIS SUBROUTINE AND EXECUTE IT AGAIN.
      IF(TIMES.EQ.0)GO TO 800
      SECTOR=0
      GO TO 100
```

```
C
800     WRITE(6,920)
920     FORMAT(2(/),8X,"CURRENT OVERLOAD TABLE",2(/))
        WRITE(6,921)((OVLTAB(I,J),J=1,4),I=1,16)
921     FORMAT((5X,4(3X,I3)))
        RETURN
        END
```

```
C
C*****THE SUBROUTINE, INTSERV, WOULD NORMALLY BE THE INTERRUPT
C*HANDLING SUBROUTINE IN A REAL OPERATING SYSTEM.  FOR THE
C*PURPOSE OF SIMULATION, THE OVERLOADED SECTOR'S  NUMBER IS
C*PASSED TO THIS SUBROUTINE VIA THE VARIABLE OVLSECT AND THE
C*SUBROUTINE PROCEEDS TO PROCESS THE OVERLOAD, FIND A SECTOR
C*WHOSE AZIMUTH COVERAGE CAN BE EXPANDED SO THAT THE OVERLOADED
C*SECTOR'S COVERAGE CAN BE SHRUNK, AND IT ALSO CALLS SUBROUTINE
C*FSTRAM SO THAT THE RAM LOGIC ARRAYS CAN BE REFORMATTED.
C
C
      SUBROUTINE INTSERV(OVLSECT,OVLTAB)
      IMPLICIT INTEGER(A-Z)
      DIMENSION OVLTAB(16,4)
C
C
C*****THE FOLLOWING IS A MESSAGE TO INDICATE THAT THE INTERRUPT
C*HANDLING SUBROUTINE HAS BEEN CALLED.  IT ALSO TELLS WHICH
C*SECTOR HAS BEEN OVERLOADED.
      WRITE(6,940)OVLSECT
940   FORMAT(2(/),5X,"EMERGENCY!!! SECTOR",1X,I2,1X,"IS
     $OVERLOADED",2(/))
C
      OVLTAB(OVLSECT,1)=1
C*****SET THE OVERLOADED SECTOR'S OVERLOAD FLAG TO ONE.
      CUROVL=OVLSECT
C*****SET THE CURRENT OVERLOAD VARIABLE TO THE OVERLOADED SECTOR'S
C*NUMBER.
      IF (OVLTAB(CUROVL,2).EQ.5)GO TO 777
C*****IF THE SECTOR IS ALREADY IN OVERLOAD STATUS FIVE THEN
C*NOTHING ELSE CAN BE DONE.
      IF (OVLTAB(CUROVL,2).EQ.4) GO TO 102
C
C*****IF THE SECTOR IS CURRENTLY IN OVERLOAD STATUS FOUR ITS
C*AZIMUTH COVERAGE CANNOT BE SHRUNK FURTHER BUT IT CAN BE PUT
C*INTO OVERLOAD STATUS FIVE WHICH INSTITUTES A FREQUENCY
C*STRIPPING OVERLOAD REDUCTION SCHEME VIA A NEW RAM LOGIC ARRAY.
C
      GO TO 103
C
C*****INCREASE THIS SECTOR'S OVERLOAD STATUS TO THE MOST SEVERE
C*CONDITION AND SET THE VARIABLES EXPAND AND CUROVL TO THE
C*SECTOR'S NUMBER.  THIS IS DONE TO INFORM SUBROUTINE FSTRAM
C*THAT THIS SECTOR'S AZIMUTH COVERAGE REMAINS UNCHANGED AND
C*FREQUENCY STRIPPING IS TO BE USED.
C
102   OVLTAB(CUROVL,2)=5
      EXPAND=CUROVL
      SECTOR=CUROVL
```

```
      GO TO 777
C
C*****IF THE OVERLOADED SECTOR'S PREVIOUS OVERLOAD STATUS WAS
C*LESS THAN FOUR THE VARIABLE OVLSTAT IS INCREMENTED TO THE NEXT
C*LEVEL OF OVERLOAD AND A SEARCH IS BEGUN TO FIND A SECTOR TO
C*EXPAND.
C
103   OVLTAB(CUROVL,2)=OVLTAB(CUROVL,2)+1
      SECTOR=1
C
C*****CHECK FOR A SECTOR WHOSE OVERLOAD FLAG IS NOT SET.
C
C
105   IF (OVLTAB(SECTOR,1).NE.0) GO TO 104
C
C*****A TEST MUST ALSO BE MADE TO INSURE THAT THE SECTOR FOUND
C*IS IN A NORMAL CONDITION,I.E.,NOT ONE WHICH IS ALREADY
C*EXPANDED.
C
      IF(OVLTAB(SECTOR,2).NE.1)GO TO 104
C*****WHEN AN EXPANDABLE SECTOR HAS BEEN FOUND SET EXPAND
C*EQUAL TO THAT SECTOR'S NUMBER.
      EXPAND=SECTOR
C
C*****IF THE OVERLOADED SECTOR IS NOT ONE WHICH WAS PREVIOUSLY
C*EXPANDED, SKIP THE NEXT INSTRUCTIONS.  OTHERWISE, EXCHANGE
C*EXPANDER POINTERS BETWEEN THE PREVIOUSLY EXPANDED SECTOR AND
C*THE NEW ONE TO BE EXPANDED.
C
      IF (OVLTAB(CUROVL,2).NE.1)GO TO 201
      OVLTAB(SECTOR,3)=OVLTAB(CUROVL,3)
C*****RESET THE PREVIOUSLY EXPANDED SECTOR'S EXPANDER POINTER
      OVLTAB(CUROVL,3)=0
C*****SET THE OVERLOAD STATUS FOR THE SECTOR TO BE EXPANDED
C*TO ZERO, INDICATING AN EXPANDED SECTOR.
      OVLTAB(SECTOR,2)=0
      GO TO 106
C
C*****IF THE SECTOR WHICH IS OVERLOADED WAS NOT PREVIOUSLY
C*EXPANDED THEN JUST SET THE EXPANDABLE SECTOR'S EXPANDER
C*POINTER TO THE OVERLOADED SECTOR'S NUMBER AND SET ITS
C*OVERLOAD STATUS VARIABLE TO ZERO.
C
201   OVLTAB(SECTOR,3)=CUROVL
      OVLTAB(SECTOR,2)=0
      GO TO 106
C
C*****NEXT,TEST TO SEE IF ALL SECTORS HAVE BEEN CHECKED IN THE
C*SEARCH FOR AN EXPANDABLE SECTOR.  IF NOT,INCREMENT THE
C*SECTOR COUNTER AND CHECK THE NEXT SECTOR.  IF ALL SECTORS
```

```
C*HAVE BEEN CHECKED, INDICATE THAT FACT BY OUTPUTTING ONLY THE
C*CURRENT OVERLOAD TABLE.  THIS OUTPUT DIFFERS FROM THE OUTPUT
C*WHEN A SECTOR TO EXPAND HAS BEEN FOUND IN THAT IT LACKS THE
C*MESSAGE CONTAINING THE OVERLOADED SECTOR'S NUMBER, THE
C*EXPANDED SECTOR'S NUMBER AND THE CALL TO FSTRAM.
C
104   IF (SECTOR.GE.16) GO TO 700
      SECTOR=SECTOR+1
      GO TO 105
106   SECTOR=CUROVL
777   WRITE(6,901)
901   FORMAT(3(/),5X,"AT THIS POINT, FSTRAM WOULD BE CALLED AND")
      WRITE(6,904)
904   FORMAT(5X,"WHEN CONTROL WAS PASSED BACK  IT WOULD")
      WRITE(6,905)
905   FORMAT(5X,"RETURN CONTROL BACK TO NOPPROS")
      WRITE(6,910)CUROVL
910   FORMAT(2(/),5X,"THE OVERLOADED SECTOR IS",1X,I2)
      WRITE(6,911)EXPAND
911   FORMAT(1(/),5X,"THE SECTOR TO BE EXPANDED IS",1X,I2,2(/))
700   WRITE(6,930)
930   FORMAT(2(/),8X,"CURRENT OVERLOAD TABLE",2(/))
      WRITE(6,931)((OVLTAB(I,J),J=1,4),I=1,15)
931   FORMAT((5X,4(3X,I3)))
      RETURN
      END
```

```
C*****************************************************
C*****************************************************
C*********************PROGRAM TEST********************
C*****************************************************
C*****************************************************
C
C
C
C*****PROGRAM TEST IS A DRIVER TO EXERCISE THE SUBROUTINES
C*FSTRAM AND SECRAM WHICH ARE THE SUBROUTINES THAT CREATE
C*THE LOGIC ARRAY PROGRAMS FOR THE FIRST LEVEL AND SECTOR
C*RAMS.  IT ALSO USES A SUBROUTINE DATAOUT WHICH SIMULATES
C*THE EXTRACTION OF DATA FROM AN INPUT DATA STREAM AND ITS
C*PROPER ROUTING VIA THE LOGIC ARRAYS.  THE INITIAL OVER-
C*LOAD TABLE WHICH IT USES IS ONE WHICH REFLECTS A NORMAL
C*STATE WITH ALL SECTORS IN THEIR NORMAL AZIMUTH COVERAGE.
C*THE OTHER ARRAYS WHICH IT USES ARE: RAMTAB-THE FIRST
C*LEVEL LOGIC ARRAY PROGRAM: SRAMTAB-THE SECTOR LOGIC
C*ARRAY PROGRAMS: AND DATABUF-A TABLE SIMULATING AN INPUT
C*DATA STREAM OF PULSE DESCRIPTOR WORDS.
C
C
C
      PROGRAM TEST(INPUT,OUTPUT,TAPE6=OUTPUT)
      IMPLICIT INTEGER(A-Z)
      DIMENSION OVLTAB(16,4),RAMTAB(256),SRAMTAB(1024),
     $DATABUF(16,4)
C
C
C*****THE FOLLOWING STATEMENTS LOAD THE OVERLOAD TABLE
TO A REFLECT A NORMAL STATE.
C
C
      DO 100 I=1,16
100   OVLTAB(I,1)=0
      DO 101 J=1,16
101   OVLTAB(J,2)=1
      DO 102 K=1,16
102   OVLTAB(K,3)=0
      DO 103 L=1,16
103   OVLTAB(L,4)=0
      WRITE(6,900)
900   FORMAT(5(/),28X,"INITIAL OVERLOAD TABLE",2(/))
      WRITE(6,901)((OVLTAB(M,N),N=1,4),M=1,16)
901   FORMAT((25X,4(3X,I3)))
C
C
C*****THE VARIABLE EXPAND IS SET TO REFLECT THE FACT THAT
C*SECTOR 16 IS TO BE EXPANDED AND THE VARIABLE CUROVL IS
C*SET TO 1 TO REFLECT THE FACT THAT SECTOR 1 IS OVER-
```

```
C*LOADED.  SETTING THESE VARIABLES TO THE ABOVE VALUES
C*ALLOWS THE LOGIC ARRAY FORMATTING SUBROUTINES TO FORMAT
C*THE ARRAYS FOR A NORMAL STATE.  THIS IS DONE FOR TEST
C*PURPOSES HERE BUT WOULD ALSO BE DONE NORMALLY TO SET
C*THE LOGIC ARRAYS TO THEIR INITIAL CONDITIONS.
C
C
      EXPAND=16
      OVROVL=1
C
C
C*****CALL THE SUBROUTINE FSTRAM WITH THE FOLLOWING
C*INPUT AND OUTPUT ARGUEMENTS: EXPAND-INPUT ARG TO
C*FSTRAM TO INDICATE THE SECTOR TO BE EXPANDED:
C*OVROVL-INPUT ARG TO FSTRAM TO INDICATE THE OVER-
C*LOADED SECTOR: RAMENT-OUTPUT ARG FROM FSTRAM,USED
C*BY SECRAM AS A POINTER TO THE SECTOR CURRENTLY
C*BEING REFORMATTED: WIDTH-OUTPUT ARG FROM FSTRAM
C*USED BY SECRAM TO INDICATE THE SECTOR'S WIDTH:
C*RAMTAB-OUTPUT ARG FROM FSTRAM WHICH IS THE FIRST
C*LEVEL LOGIC ARRAY PROGRAM: SECTOR-INPUT/OUTPUT ARG
C*TO/FROM FSTRAM WHICH IS A GENERAL PURPOSE POINTER:
C*OVLTAB-INPUT ARG TO FSTRAM INDICATING THE OVERALL
C*STATUS OF THE SYSTEM: AND SRAMTAB WHICH IS USED BY
C*SECRAM AND IS INCLUDED HERE ONLY FOR COMPLETENESS.
C
C
      CALL FSTRAM(EXPAND,OVROVL,RAMENT,WIDTH,RAMTAB,SECTOR,
     $OVLTAB,SRAMTAB)
C
C*****FSTRAM CALLS SECRAM AND WHEN ALL THE NECESSARY
C*SECTORS HAVE BEEN REFORMATTED IT RETURNS TO PROGRAM
C*TEST AT WHICH POINT THE LOGIC ARRAYS ARE PRINTED OUT.
C
      WRITE(6,701)
701   FORMAT(5(/),35X,"SRAMTAB FOLLOWS",2(/))
      WRITE(6,511)(SRAMTAB(I),I=1,1024)
511   FORMAT((10X,10(3X,I2)))
C
C*****NOW CALL THE SUBROUTINE DATAOUT WHICH WILL READ
C*DATA (STIMULATING A REAL INPUT DATA STREAM) FROM A
C*TABLE CALLED DATABUF AND PRINT OUT THAT DATA WITH AN
C*INDICATION OF ITS PROPER DESTINATION.  IT USES THE
C*LOGIC ARRAYS TO MAKE ITS ROUTING DECISIONS.
C
      CALL DATAOUT(RAMTAB,SRAMTAB,DATABUF)
      STOP
      END
```

```
C
C
C*****THE SUBROUTINE FSTRAM DOES THE REFORMATTING OF
C*THE FIRST LEVEL LOGIC ARRAY PROGRAM.  ITS INPUT AND
C*OUTPUT ARGUEMENTS ARE EXPLAINED IN THE LISTING FOR
C*PROGRAM TEST.
C
      SUBROUTINE FSTRAM(EXPAND,CUROVL,RAMENT,WIDTH,RAMTAB,SECTOR,
     $OV_TAB,SRAMTAB)
      IMPLICIT INTEGER(A-Z)
C
C*****THE NEXT TWO STATEMENTS INITIALIZE THE FIRST LEVEL
C*LOGIC ARRAY TO ZERO.  ****WARNING-THIS IS FOR THIS TEST
C*ROUTINE ONLY.  IN A NORMAL SITUATION FSTRAM DOES NOT RE-
C*FORMAT THE ENTIRE LOGIC ARRAY-IT ONLY WORKS ON THE
C*SECTORS BETWEEN THE ONE TO BE EXPANDED AND THE ONE WHICH
C*IS OVERLOADED. LEAVING THESE STATEMENTS IN FOR NORMAL
C*OPERATION WOULD CAUSE SOME OF THE ARRAY TO ALWAYS BE SET
C*TO ZERO.*****
C
      DIMENSION OVLTAB(16,4),RAMTAB(256),SRAMTAB(1024)
      DO 111 I=1,256
111   RAMTAB(I)=0
C
C*****THESE NEXT 5 STATEMENTS SET UP THE VARIABLES FSTSECT
C*AND LSTSECT.  FSTRAM REFORMATS THE SECTORS BETWEEN
C*FSTSECT AND LSTSECT IN ASCENDING ORDER, BEGINNING AT
C*FSTSECT.
C
      IF (EXPAND.GT.CUROVL)GO TO 201
      FSTSECT=EXPAND
      LSTSECT=CUROVL
      GO TO 102
201   FSTSECT=CUROVL
      LSTSECT=EXPAND
C
C*****STARTING WITH SECTOR ONE THE VARIABLE RAMENT IS
C*CALCULATED.  RAMENT INDICATES HOW MANY AZIMUTH INCREMENTS
C*ARE CONTAINED IN THE SECTORS PRECEEDING FSTSECT. ONE
C*AZIMUTH INCREMENT IS APPROXIMATELY EQUAL TO 1.4 DEGREES
C*AND THE WIDTH OF A SECTOR EXPRESSED IN AZIMUTH INCREMENTS
C*VARIES ACCORDING TO THE SECTOR'S OVERLOAD STATUS.
C*TO CALCULATE RAMENT THESE STATEMENTS SEQUENCE THROUGH THE
C*OVERLOAD TABLE AND INCREMENT IT ACCORDING TO THE VALUE OF
C*OVLSTAT.  FOR OVLSTAT=0,RAMENT IS INCREMENTED BY 20; FOR
C*OVLSTAT=1, RAMENT IS INCREMENTED BY 16; FOR OVLSTAT=2,
C*RAMENT IS INCREMENTED BY 12; FOR OVLSTAT=3, RAMENT IS
C*INCREMENTED BY 8; AND FOR OVLSTAT=4 OR 5, RAMENT IS
C*INCREMENTED BY 4.
```

```
C
102   SECTOR=1
      RAMENT=1
206   IF(SECTOR.NE.FSTSECT)GO TO 202
      GO TO 303
202   IF(OVLTAB(SECTOR,2).NE.0)GO TO 203
      RAMENT=RAMENT+20
      GO TO 207
203   IF(OVLTAB(SECTOR,2).NE.1)GO TO 204
      RAMENT=RAMENT+16
      GO TO 207
204   IF(OVLTAB(SECTOR,2).NE.2)GO TO 205
      RAMENT=RAMENT+12
      GO TO 207
205   IF(OVLTAB(SECTOR,2).NE.3)GO TO 206
      RAMENT=RAMENT+8
      GO TO 207
206   RAMENT=RAMENT+4
C
C*****THE FOLLOWING GROUP OF STATEMENTS ARE USED TO
C*CALCULATE THE WIDTH, IN AZIMUTH INCREMENTS, OF THE
C*SECTOR CURRENTLY BEING REFORMATTED.  THEY USE THE SAME
C*LOGIC AS THE STATEMENTS WHICH CALCULATED RAMENT.
C
207   SECTOR=SECTOR+1
      IF(SECTOR.LE.16)GO TO 208
303   IF(OVLTAB(SECTOR,2).NE.0)GO TO 210
      WIDTH=20
      GO TO 214
210   IF(OVLTAB(SECTOR,2).NE.1)GO TO 211
      WIDTH=16
      GO TO 214
211   IF(OVLTAB(SECTOR,2).NE.2)GO TO 212
      WIDTH=12
      GO TO 214
212   IF(OVLTAB(SECTOR,2).NE.3)GO TO 213
      WIDTH=8
      GO TO 214
213   WIDTH=4
C
C*****NOW THAT THE WIDTH OF THIS SECTOR IS KNOWN THE
C*LAST LOGIC ARRAY ENTRY TO BE REFORMATTED CAN BE
C*CALCULATED.  THAT VALUE IS SAVED AS VARIABLE LSTENT.
C*THE FIRST LOGIC ARRAY ELEMENT TO BE REFORMATTED IS
C*RAMTAB(RAMENT+1).
C
214   LSTENT=RAMENT+WIDTH
C
C*****THE CONTENT OF EACH LOGIC ARRAY ELEMENT IS THE
C*SECTOR NUMBER TO WHICH THIS PARTICULAR AZIMUTH ELEMENT
```

```
C*HAS BEEN ASSIGNED (FIRST LEVEL LOGIC ARRAY ELEMENT=
C*AZIMUTH INCREMENT=1.4 DEGREES).  THE SECTOR NUMBERS ARE
C*EXPRESSED AS 16 BIT QUANTITIES WITH ONE BIT FOR EACH
C*SECTOR.  THEY ARE CALCULATED BY THE EQUATION:
C*"RAMCONT=2**SECTOR-1".  THE VALUE OF RAMCONT IS THEN STORED
C*IN EACH LOGIC ARRAY ELEMENT ASSIGNED TO A PARTICULAR
C*SECTOR.
105     RAMCONT=2**(SECTOR-1)
        RAMTAB(RAMENT)=RAMCONT
104     RAMENT=RAMENT+1
        IF(RAMENT.NE.LSTENT)GO TO 105
C
C*****AFTER EACH OF THE SECTORS IN THE FIRST LEVEL RAM HAS
C*BEEN REFORMATTED THEN THE SUBROUTINE SECRAM HAS TO BE
C*CALLED TO REFORMAT THE SECOND LEVEL LOGIC ARRAY RAM FOR
C*THAT SECTOR.  EACH OF THE SECOND LEVEL LOGIC ARRAYS HAS
C*THE RESPONSIBILITY OF ASSIGNING FOUR MICROPROCESSORS
C*TO THE AZIMUTH COVERAGE ALLOCATED TO A SECTOR.  THE
C*INPUT AND OUTPUT ARGUEMENTS FOR SECRAM ARE DESCRIBED
C*IN THAT SUBROUTINE'S COMMENTS.
C
        CALL SECRAM (RAMENT,WIDTH,OVLTAB,SRAMTAB,SECTOR)
C
C*****GO BACK AND REFORMAT THE NEXT SECTOR UNLESS THE ONE
C*JUST FINISHED WAS THE LAST SECTOR.
C
        IF(SECTOR.EQ.LSTSECT)GO TO 500
        SECTOR=SECTOR+1
        GO TO 303
C
C*****PRINT OUT THE NEW FIRST LEVEL LOGIC ARRAY.
C
500     WRITE(6,700)
700     FORMAT(5(/),35X,"RAMTAB FOLLOWS",2(/))
        WRITE(6,501)(RAMTAB(I),I=1,256)
501     FORMAT((2X,8(3X,I5)))
666     RETURN
        END
```

```
C
C
C*****THE SUBROUTINE SECRAM IS RESPONSIBLE FOR REFORMATTING THE
C*SECOND LEVEL (SECTOR LEVEL) LOGIC ARRAYS.  EACH SECTOR HAS 4
C*MICROPROCESSORS AVAILABLE TO ASSIGN AND NORMALLY EACH ONE IS
C*ASSIGNED ONE FOURTH OF THE SECTOR'S AZIMUTH COVERAGE.  AS THE
C*SECTORS'S AZIMUTH COVERAGE CHANGES DUE TO OVERLOADS OR
C*REDUCTION OF OVERLOADS THE MICROPROCESSOR ASSIGNMENTS MUST
C*ALSO CHANGED.  WHEN A SECTOR'S AZIMUTH COVERAGE IS LESS THAN
C*12 AZIMUTH INCREMENTS (16.8 DEGREES) IT IS NO LONGER DIVIDED
C*INTO FOURTHS.  IF THE COVERAGE IS 8 INCREMENTS THEN 2
C*PROCESSORS ARE ASSIGNED TO EACH HALF OF THE SECTOR, AND IF
C*THE COVERAGE IS 4 INCREMENTS THEN ALL PROCESSORS ARE ASSIGNED
C*THE SAME AZIMUTH COVERAGE.  IN EITHER OF THE ABOVE CASES,
C*FREQUENCY IS USED TO HELP DECIDE HOW TO ROUTE PULSES TO THE
C*PROCESSORS.
C
C
      SUBROUTINE SECRAM (RAMENT,WIDTH,OVLTAB,SRAMTAB,SECTOR)
      IMPLICIT INTEGER(A-Z)
      DIMENSION SRAMTAB(1024),RAMTAB(256),OV_TAB(16,4)
C
C*****THE CALLING ARGUEMENTS FOR SECRAM ARE AS FOLLOWS:
C*RAMENT-INPUT ARGUEMENT WHICH TELLS SECRAM WHERE FSRRAM
C*LEFT OFF IN ITS REFORMATTING EFFORT: WIDTH-INPUT ARG
C*WHICH TELLS SECRAM HOW WIDE THE SECTOR IS: OVLTAB-INPUT
C*ARG WHICH PROVIDES SECRAM WITH EACH SECTOR'S OVERLOAD
C*STATUS: SRAMTAB-OUTPUT ARG WHICH IS THE NEW LOGIC ARRAY:
C*AND SECTOR-INPUT ARG WHICH IS A GENERAL PURPOSE POINTER.
C
C
C*****THE VALUE OF RAMENT IS DECREMENTED TO THE LAST FIRST
C*LEVEL RAM ENTRY WHICH WAS REFORMATTED.  SECRAM WILL WORK
C*BACKWARDS FROM THERE IN ITS REFORMATTING EFFORT.  RAMENT
C*WILL BE RESTORED BEFORE EXITING THIS SUBROUTINE.
C
      RAMENT=RAMENT-1
C*****SET SRAMENT TO THE VALUE OF RAMENT FOR INTERNAL
C*SUBROUTINE USE.
C
      SRAMENT=RAMENT
C
C*****DIVIDE THE SECTOR'S WIDTH BY FOUR SO THAT EACH
C*PROCESSOR WILL PROCESS ONE FOURTH OF THE SECTOR'S
C*AZIMUTH COVERAGE.  IF, HOWEVER, BWIDTH IS LESS THAN
C*3 THEN FREQUENCY WILL BE REQUIRED IN ADDITION TO ADA
C*FOR THE DATA STREAM ROUTING.
C
      BWIDTH=WIDTH/4
```

```
      IF(BWIDTH.EQ.2)GO TO 101
      IF(BWIDTH.EQ.1)GO TO 102
C
C*****FOR THE CASES WHEN THE SECTOR'S AZIMUTH COVERAGE IS
C*12 INCREMENTS OR GREATER AND FREQUENCY IS TO BE IGNORED
C*THE MICROPROCESSOR (BIN) ASSIGNMENTS MUST BE REPEATED 4
C*TIMES.  THE ADDRESS INPUT TO THIS LOGIC ARRAY CONSISTS OF
C*THE 8 BITS OF ADA AND THE 2 MOST SIGNIFICANT BITS OF THE
C*16 FREQUENCY BITS IN A PULSE DESCRIPTOR WORD.  THE 2
C*FREQUENCY BITS ARE ALSO THE 2 MSB'S OF THE LOGIC ARRAY
C*ADDRESS.  TO IGNORE FREQUENCY, THE BIN ASSIGNMENTS ARE
C*ARE PLACED IN THE ADDRESSES SPECIFIED BY ADA AND EACH
C*POSSIBLE COMBINATION OF THE 2 FREQ BITS (01,10,11,&00).
C
C*****THE VARIABLE TIMES COUNTS THE NUMBER OF TIMES THAT
C*THE BIN ASSIGNMENTS HAVE LEFT TO BE REPEATED AND THE
C*VARIABLE BIN CONTROLS WHICH MICROPROCESSOR IS TO BE
C*ASSIGNED (BIN'S POSSIBLE VALUES ARE 0,1,2,&3).
C
C*****THE CALCULATION OF EACH ARRAY ELEMENT'S CONTENTS NOW
C*PROCEEDS IN THE SAME FASHION AS FOR THE FIRST LEVEL ARRAY.
C
      TIMES=4
403   BIN=3
104   BINCONT=2**BIN
C
C*****THE FIRST TIME THROUGH, SRAMENT, WHICH IS THE POINTER INTO
C*THE SECTOR RAM LOGIC ARRAYS, IS SET TO CONSIDER ADA AS THE
C*ADDRESSING VARIABLE FOR DATA EXTRACTION(BIN SELECTION).  THIS
C*SECTION OF THE TABLE WOULD BE ADDRESSED WHEN THE 2 FREQUENCY
C*BITS WERE 00.
C
      WIDCNT=SRAMENT-BWIDTH
105   SRAMTAB(SRAMENT)=BINCONT
      SRAMENT=SRAMENT-1
      IF(SRAMENT.NE.WIDCNT)GO TO 105
      IF(BIN.EQ.0)GO TO 106
      BIN=BIN-1
      GO TO 104
C
C*****THESE NEXT 13 STATEMENTS ALLOW FOR THE REPETITION OF THE
C*LOGIC ARRAY FORMAT CREATED FOR FREQ=00 IN THE ADDRESSES WHERE
C*FREQ=01,10,&11.  THE FREQUENCY BITS ARE ATTACHED TO RAM
C*ADDRESSES 2**8 AND 2**9.
C
106   TIMES=TIMES-1
      IF(TIMES.EQ.3)GO TO 200
      GO TO 201
200   SRAMENT=(2**8)+SRAMENT
      GO TO 403
```

```
201     IF(TIMES.EQ.2)GO TO 202
        GO TO 203
202     SRAMENT=(2**9)+RAMENT
        GO TO 403
203     IF(TIMES.EQ.1)GO TO 204
        GO TO 510
204     SRAMENT=(2**8)+(2**9)+RAMENT
        GO TO 403
C
C*****THIS NEXT SECTION OF CODE (LABELED 101) ALLOWS FOR THE CASE
C*WHEN THE AZIMUTH SECTOR BEING REFORMATTED IS ONLY 8 AZIMUTH
C*INCREMENTS (11.25 DEGREES) WIDE AND CANNOT BE DIVIDED INTO 4
C*BINS, ONE FOR EACH MICROPROCESSOR.  THE REASON IT CANNOT BE
C*DIVIDED IS THAT WOULD CREATE A BIN WIDTH OF 2.8 DEGREES AND THE
C*AOA OF A RECEIVED PULSE TRAIN MAY NOT REMAIN IN THAT NARROW A
C*SECTOR LONG ENOUGH TO COMPLETE THE ECM PROCESSING.  THE STATEMENTS
C*UNDER LABEL 101 ASSIGN BINS 0 AND 1 TO THE ADRESSES SPECIFIED
C*BY AOA AND FREQ=01: AND ASSIGNS BINS 2 AND 3 TO THE ADDRESSES
C*SPECIFIED BY AOA AND FREQ=00.  THE  TATEMENTS UNDER LABEL 107
C*ASSIGN BINS 0 AND 1 TO ADDRESSES SPECIFIED BY AOA AND FREQ=10:
C* AND ASSIGN BINS 2 AND 3 TO ADDRESSES SPECIFIED BY AOA AND FREQ=11.
C
101     SRAMENT=(2**8)+RAMENT
        SRAMTAB(SRAMENT)=1
        SRAMENT=SRAMENT-1
        SRAMTAB(SRAMENT)=2
        SRAMENT=RAMENT
        SRAMTAB(SRAMENT)=4
        SRAMENT=SRAMENT-1
        SRAMTAB(SRAMENT)=8
107     SRAMENT=(2**9)+RAMENT
        SRAMTAB(SRAMENT)=1
        SRAMENT=SRAMENT-1
        SRAMTAB(SRAMENT)=2
        SRAMENT=(2**8)+(2**9)+RAMENT
        SRAMTAB(SRAMENT)=4
        SRAMENT=SRAMENT-1
        SRAMTAB(SRAMENT)=8
        GO TO 510
C
C*****THE STATEMENTS UNDER LABEL 102 ARE CALLED WHEN THE SECTOR
C*BEING REFORMATTED IS ONLY ONE AZIMUTH INCREMENT (5.625 DEGREES)
C*WIDE.  THEY ASSIGN ONE MICROPROCESSOR (BIN) TO ADRESSES
C*SPECIFIED BY AOA AND EACH OF THE 4 COMBINATIONS OF THE 2 FREQ
C*BITS.  TO AMPLIFY THE SITUATION, EACH MICROPROCESSOR WILL NOW
C*BE RESPONSIBLE FOR PROCESSING PULSES WITHIN A 5.625 DEGREE
C*AZIMUTH SECTOR AND ONE FOURTH OF THE FREQUENCY SPECTRUM.
C
102     SRAMENT=(2**9)+RAMENT
        SRAMTAB(SRAMENT)=1
```

```
      SRAMENT=(2**9)+(2**8)+RAMENT
      SRAMTAB(SRAMENT)=2
      SRAMENT=(2**8)+RAMENT
      SRAMTAB(SRAMENT)=4
      SRAMENT=RAMENT
      SRAMTAB(SRAMENT)=8
C**** RESTORE RAMENT TO ITS ORIGINAL VALUE.
510   RAMENT=RAMENT+1
      RETURN
      END
```

```
C
C****THE SUBROUTINE DATAOUT SIMULATES THE NORMAL STEERING OF THE
C*PULSE DESCRIPTOR WORDS TO THEIR PROPER LOCATION.  IT USES THE
C*TWO LEVELS OF LOGIC ARRAYS TO STEER THE DATA.  DATABUF IS AN
C*ARRAY CONTAINING THE TYPE OF DATA WHICH WOULD NORMALLY BE INPUT
C*TO THE MICROPROCESSOR NETWORK.
C
C
      SUBROUTINE DATAOUT(RAMTAB,SRAMTAB,DATABUF)
      IMPLICIT INTEGER(A-Z)
      DIMENSION RAMTAB(256),SRAMTAB(1024),DATABUF(16,4)
C
C****THE NEXT GROUP OF STATEMENTS LOADS THE INPUT DATA ARRAY WITH
C*16 PULSE DESCRIPTOR WORDS (EXCLUDING PULSE WIDTH AND PULSE
C*AMPLITUDE) FROM SOME FICTICIOUS RADARS.  DATABUF'S FIRST COLUMN
C*CONTAINS AOA EXPRESSED IN AZIMUTH INCREMENTS OF 1.4 DEGREES.
C*DATABUF'S SECOND COLUMN CONTAINS FREQUENCY EXPRESSED IN
C*INCREMENTS OF .5 MHZ. THE THIRD COLUMN CONTAINS TIME OF ARRIVAL
C*EXPRESSED IN NANOSECONDS FROM TIME ZERO.  THE FOURTH COLUMN
C*CONTAINS A CONCATONTION OF AOA WITH THE 2 MOST SIGNIFICANT BITS
C*OF FREQUENCY (FOR USE IN ADDRESSING THE SECOND LEVEL LOGIC
C*ARRAYS).
C
      DATABUF(1,1)=7
      DATABUF(1,2)=1000
      DATABUF(1,3)=2000
      DATABUF(1,4)=7
      DATABUF(2,1)=103
      DATABUF(2,2)=18000
      DATABUF(2,3)=4000
      DATABUF(2,4)=359
      DATABUF(3,1)=7
      DATABUF(3,2)=6000
      DATABUF(3,3)=6000
      DATABUF(3,4)=7
      DATABUF(4,1)=7
      DATABUF(4,2)=1000
      DATABUF(4,3)=8000
      DATABUF(4,4)=7
      DATABUF(5,1)=132
      DATABUF(5,2)=54000
      DATABUF(5,3)=10000
      DATABUF(5,4)=960
      DATABUF(6,1)=128
      DATABUF(6,2)=3000
      DATABUF(6,3)=12000
      DATABUF(6,4)=128
      DATABUF(7,1)=7
      DATABUF(7,2)=1000
```

```
      DATABUF(7,3)=14000
      DATABUF(7,4)=7
      DATABUF(8,1)=103
      DATABUF(8,2)=18000
      DATABUF(8,3)=16000
      DATABUF(8,4)=359
      DATABUF(9,1)=7
      DATABUF(9,2)=5000
      DATABUF(9,3)=18000
      DATABUF(9,4)=7
      DATABUF(10,1)=7
      DATABUF(10,2)=1000
      DATABUF(10,3)=20000
      DATABUF(10,4)=7
      DATABUF(11,1)=192
      DATABUF(11,2)=54000
      DATABUF(11,3)=22000
      DATABUF(11,4)=960
      DATABUF(12,1)=128
      DATABUF(12,2)=3000
      DATABUF(12,3)=24000
      DATABUF(12,4)=128
      DATABUF(13,1)=7
      DATABUF(13,2)=1000
      DATABUF(13,3)=26000
      DATABUF(13,4)=7
      DATABUF(14,1)=103
      DATABUF(14,2)=16000
      DATABUF(14,3)=28000
      DATABUF(14,4)=359
      DATABUF(15,1)=7
      DATABUF(15,2)=6000
      DATABUF(15,3)=30000
      DATABUF(15,4)=7
      DATABUF(16,1)=7
      DATABUF(16,2)=1000
      DATABUF(16,3)=32000
      DATABUF(16,4)=7
C
C*****PRINT OUT THE DATA TO BE INPUT TO THE PROCESSOR NETWORK.
C
      WRITE(6,930)
930   FORMAT(2(/),28X,"INITIAL DATABUF TABLE",2(/))
      WRITE(6,931)((DATABUF(O,P),P=1,4),O=1,15)
931   FORMAT((20X,4(3X,I5)))
C
C*****THE STATEMENTS FOLLOWING LABEL 100 SEARCH THE FIRST LEVEL
C*LOGIC ARRAY TO FIND THE PROPER SECTOR FOR THE DATA IN THE
C*INPUT DATA TABLE>  AOA IS USED FOR THE ADDRESS INTO THE LOGIC
C*ARRAY AND WHAT IS EXTRACTED IS* 2**(SECTOR-1).  THAT DATA IS
```

```
C*THEN DIVIDED BY 2**(SECTOR-1) UNTIL THE PROPER VALUE FOR SECTOR
C*IS FOUND, I.E.,TRY=1.
C
      DO 200 I=1,16
      SECTOR=1
100   TRY=RAMTAB(DATABUF(I,1))
      TRY=TRY/2**(SECTOR-1)
      IF(TRY.EQ.1)GO TO 800
      SECTOR=SECTOR+1
      GO TO 100
C
C*****PRINT OUT THE SECTOR NUMBER WHICH WAS SELECTED.
C
800   WRITE(6,900)I,SECTOR
900   FORMAT(2(/),20X,"THE DATA IN DATABUF LINE",2X,I2,
     $2X,"BELONGS TO SECTOR",2X,I2)
      WRITE(6,901)
901   FORMAT(20X,"BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS")
C
C*****THE FOLLOWING STATEMENTS SELECT THE PROPER MICROPROCESSOR
C*TO SEND THE DATA TO IN MUCH THE SAME MANNAR AS THE SECTOR

C*SELECTION.
C
      BIN=1
101   STRY=SRAMTAB(DATABUF(I,4))*2
      STRY=STRY/2**BIN
      IF(STRY.EQ.1)GO TO 801
C
C*****PRINT OUT THE BIN NUMBER SELECTED.
C
      BIN=BIN+1
      GO TO 101
801   WRITE(6,902)BIN,SECTOR
902   FORMAT(19X,"BIN",2X,I1,2X,"OF SECTOR",
     $2X,I2,2X,"WAS CHOSEN")
C
C*****THE FOLLOWING STATEMENTS SCALE FREQUENCY TO UNITS OF MHZ
C*AND PRINT OUT THE FREQUENCY AND TIME OF ARRIVAL FOR EACH PULSE
C*DESCRIPTOR WORD AS ITS PROPER DESTINATION IS SELECTED.
C
      FREQ=DATABUF(I,2)/2
      WRITE(6,903)FREQ,DATABUF(I,3)
903   FORMAT(5X,"THE EMITTERS FREQUENCY IS",2X,I5,2X,
     $"MHZ AND ITS TOA IS",2X,I5,2X,"NS",2(/))
200   CONTINUE
      RETURN
      END
```

Appendix B

This appendix is published as a separate document entitled, A Parallel Microprocessor Architecture for Electronic Countermeasures Processing - Appendix B.

Appendix C

Test Results

INITIAL OVERLOAD TABLE

0	0	8	0
0	0	9	0
0	0	8	0
0	0	9	0
0	0	7	0
0	0	10	0
1	2	0	0
1	3	0	2
1	3	0	3
1	2	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0

In the initial overload table sectors 7 and 10 are in overload state two causing sectors 5 and 6 respectively to be expanded. Sectors 8 and 9 are both in overload state three and have caused sectors 1,3,2, and 4, respectively , to be expanded. Sectors 11-16 are in normal azimuth coverage.

NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED 8 TIMES

CURRENT OVERLOAD TABLE

0	1	0	0
0	1	0	0
0	0	8	0
0	0	9	0
0	1	0	0
0	1	0	0
0	1	0	0
1	2	0	3
1	2	0	4
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0

The overloads in sectors 7 and 10 have been eliminated, sectors 8 and 9 have been reduced to an overload state of two, and sectors 1,2,5, and 6 have been changed from the expanded state to the normal state.

EMERGENCY!!! SECTOR 1 IS OVERLOADED

```
AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  1

THE SECTOR TO BE EXPANDED IS  2


   CURRENT OVERLOAD TABLE

     1      2      0      0
     0      0      1      0
     0      0      8      0
     0      0      9      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     1      2      0      3
     1      2      0      4
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0

NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED 10 TIMES

   CURRENT OVERLOAD TABLE

     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
```

Sector 1 has re-overloaded causing sector 2 to be expanded.

NORPROS was executed enough times to reduce the overloads in all sectors back to normal. This is a clean overload table, i.e., all sectors are in normal azimuth coverage.

```
EMERGENCY!!! SECTOR  1 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  1

THE SECTOR TO BE EXPANDED IS  2


   CURRENT OVERLOAD TABLE

     1      2      0      0
     0      0      1      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0

EMERGENCY!!! SECTOR 16 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 16

THE SECTOR TO BE EXPANDED IS  3
```

The overload in sector 1 was removed too soon and it re-overloads.

CURRENT OVERLOAD TABLE

1	2	0	0
0	0	1	0
0	0	16	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
1	2	0	0

Sector 16 has now overloaded as the aircraft approaches the simulated dense radio frequency environment.

NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED 5 TIMES

CURRENT OVERLOAD TABLE

1	2	0	5
0	0	1	0
0	0	16	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
1	2	0	5

There is a lull in the overloads and NORPROS executes five times. No overloads are reduced but the overload reduction count for sectors 1 and 16 has reached five.

EMERGENCY!!! SECTOR 1 IS OVERLOADED

```
AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  1

THE SECTOR TO BE EXPANDED IS  4


   CURRENT OVERLOAD TABLE

     1      3      0      5
     0      0      1      0
     0      0     16      0
     0      0      1      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     1      2      0      5

EMERGENCY!!! SECTOR 16 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 16

THE SECTOR TO BE EXPANDED IS  5
```

Sector 1 overloads again and now its in overload state three.

CURRENT OVERLOAD TABLE

1	3	0	5
0	0	1	0
0	0	16	0
0	0	1	0
0	0	16	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
1	3	0	5

The aircraft is approaching the simulated dense environment and the sectors around the nose begin to overload more severely.

EMERGENCY!!! SECTOR 15 IS OVERLOADED

AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK IT WOULD
RETURN CONTROL BACK TO NOPPROS

THE OVERLOADED SECTOR IS 15

THE SECTOR TO BE EXPANDED IS 6

CURRENT OVERLOAD TABLE

1	3	0	5
0	0	1	0
0	0	16	0
0	0	1	0
0	0	16	0
0	0	15	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
1	2	0	0
1	3	0	5

```
EMERGENCY!!! SECTOR  2 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  2

THE SECTOR TO BE EXPANDED IS  7


   CURRENT OVERLOAD TABLE

     1     3     0     5
     1     1     0     0
     0     0    16     0
     0     0     1     0
     0     0    16     0
     0     0    15     0
     0     0     1     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     1     2     0     0
     1     3     0     5

NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED  3 TIMES
```

```
   CURRENT OVERLOAD TABLE

     1      2      0      1
     1      1      0      3
     0      1      0      0
     0      1      0      0
     0      0     16      0
     0      0     15      0
     0      0      1      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     1      2      0      3
     1      2      0      1

EMERGENCY!!! SECTOR  1 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS


THE OVERLOADED SECTOR IS  1

THE SECTOR TO BE EXPANDED IS  3

   CURRENT OVERLOAD TABLE

     1      3      0      1
     1      1      0      3
     0      0      1      0
     0      1      0      0
     0      0     16      0
     0      0     15      0
     0      0      1      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     1      2      0      3
     1      2      0      1
```

There is a short lull in the overloads and NORPROS executes three times. Since the overload reduction counts in sectors 1 and 16 were already at five, their overload states were reduced one level.

The overloading continues as the aircraft approaches closer to the dense environment.

```
EMERGENCY!!! SECTOR 16 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 16

THE SECTOR TO BE EXPANDED IS  4


   CURRENT OVERLOAD TABLE

     1      3      0      1
     1      1      0      3
     0      0      1      0
     0      0     16      0
     0      0     15      0
     0      0     15      0
     0      0      1      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     0      1      0      0
     1      2      0      3
     1      3      0      1

EMERGENCY!!! SECTOR 15 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 15

THE SECTOR TO BE EXPANDED IS  8
```

```
    CURRENT OVERLOAD TABLE

      1     3     0     1
      1     1     0     3
      0     0     1     0
      0     0    16     0
      0     0    15     0
      0     0    15     0
      0     0     1     0
      0     0    15     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      1     3     0     3
      1     3     0     1

EMERGENCY!!! SECTOR  2 IS                    OVERLOADED

AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NOFPROS

THE OVERLOADED SECTOR IS  2

THE SECTOR TO BE EXPANDED IS  9

    CURRENT OVERLOAD TABLE

      1     3     0     1
      1     2     0     3
      0     0     1     0
      0     0    16     0
      0     0    15     0
      0     0    15     0
      0     0     1     0
      0     0    15     0
      0     0     2     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      0     1     0     0
      1     3     0     3
      1     3     0     1
```

```
NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED  2 TIMES

   CURRENT OVERLOAD TABLE

     1     3     0     3
     1     2     0     5
     0     0     1     0
     0     0    16     0
     0     0    16     0
     0     0    15     0
     0     0     1     0
     0     0    15     0
     0     0     2     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     1     3     0     5
     1     3     0     3

EMERGENCY!!! SECTOR  1 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  1

THE SECTOR TO BE EXPANDED IS 10
```

There is another brief lull in the overloading and NORPROS executes twice but has no effect on the overload condition of any sector.

```
   CURRENT OVERLOAD TABLE

     1     4     0     3
     1     2     0     5
     0     0     1     0
     0     0    16     0
     0     0    15     0
     0     0    15     0
     0     0     1     0
     0     0    15     0
     0     0     2     0
     0     0     1     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     1     3     0     5
     1     3     0     3

EMERGENCY!!! SECTOR 16 IS                    OVERLOADED

AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS


THE OVERLOADED SECTOR IS 16

THE SECTOR TO BE EXPANDED IS 11

   CURRENT OVERLOAD TABLE

     1     4     0     3
     1     2     0     5
     0     0     1     0
     0     0    16     0
     0     0    15     0
     0     0    15     0
     0     0     1     0
     0     0    15     0
     0     0     2     0
     0     0     1     0
     0     0    16     0
     0     1     0     0
     0     1     0     0
     0     1     0     0
     1     3     0     5
     1     4     0     3
```

```
EMERGENCY!!! SECTOR 15 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 15

THE SECTOR TO BE EXPANDED IS 12


   CURRENT OVERLOAD TABLE

      1     4     0     3
      1     2     0     5
      0     0     1     0
      0     0    16     0
      0     0    15     0
      0     0    15     0
      0     0     1     0
      0     0    15     0
      0     0     2     0
      0     0     1     0
      0     0    15     0
      0     0    15     0
      0     1     0     0
      0     1     0     0
      1     4     0     5
      1     4     0     3

EMERGENCY!!! SECTOR  2 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  2

THE SECTOR TO BE EXPANDED IS 13
```

CURRENT OVERLOAD TABLE

1	4	0	3
1	3	0	5
0	0	1	0
0	0	16	0
0	0	16	0
0	0	15	0
0	0	1	0
0	0	15	0
0	0	2	0
0	0	1	0
0	0	16	0
0	0	15	0
0	0	2	0
0	1	0	0
1	4	0	5
1	4	0	3

NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED 1 TIMES

CURRENT OVERLOAD TABLE

1	4	0	4
1	3	0	6
0	0	1	0
0	0	16	0
0	0	16	0
0	0	15	0
0	0	1	0
0	0	15	0
0	0	2	0
0	0	1	0
0	0	16	0
0	0	15	0
0	0	2	0
0	1	0	0
1	4	0	6
1	4	0	4

EMERGENCY!!! SECTOR 9 IS OVERLOADED

```
AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NOEPROS

THE OVERLOADED SECTOR IS  9

THE SECTOR TO BE EXPANDED IS 14

   CURRENT OVERLOAD TABLE

     1     4     0     4
     1     3     0     6
     0     0     1     0
     0     0    16     0
     0     0    15     0
     0     0    15     0
     0     0     1     0
     0     0    15     0
     1     1     0     0
     0     0     1     0
     0     0    15     0
     0     0    15     0
     0     0     2     0
     0     0     2     0
     1     4     0     6
     1     4     0     4

EMERGENCY!!! SECTOR 10 IS                    OVERLOADED

   CURRENT OVERLOAD TABLE

     1     4     0     4
     1     3     0     6
     0     0     1     0
     0     0    16     0
     0     0    16     0
     0     0    15     0
     0     0     1     0
     0     0    15     0
     1     1     0     0
     1     1     1     0
     0     0    16     0
     0     0    15     0
     0     0     2     0
     0     0     2     0
     1     4     0     6
     1     4     0     4
```

The aircraft makes a turn to the right and now the azimuth sectors on the left side will be exposed to the dense radio frequency environment.

This incident shows how the EXPANDR pointer is passed from one sector to another when an expanded sector overloads. Sector 9 was expanded previously by sector 2 and now that the aircraft has turned it overloads. Sector 9's EXPANDR pointer (EXPANDR=2) is passed to an available sector which can be expanded, sector 14.

Sector 10 has overloaded but there are no sectors available to expand.

EMERGENCY!!! SECTOR 12 IS OVERLOADED

CURRENT OVERLOAD TABLE

1	4	0	4
1	3	0	6
0	0	1	0
0	0	16	0
0	0	16	0
0	0	15	0
0	0	1	0
0	0	15	0
1	1	0	0
1	1	1	0
0	0	16	0
1	1	15	0
0	0	2	0
0	0	2	0
1	4	0	6
1	4	0	4

EMERGENCY!!! SECTOR 11 IS OVERLOADED

CURRENT OVERLOAD TABLE

1	4	0	4
1	3	0	6
0	0	1	0
0	0	16	0
0	0	16	0
0	0	15	0
0	0	1	0
0	0	15	0
1	1	0	0
1	1	1	0
1	1	16	0
1	1	15	0
0	0	2	0
0	0	2	0
1	4	0	6
1	4	0	4

```
NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED  1 TIMES

   CURRENT OVERLOAD TABLE

     1     4     0     5
     1     2     0     0
     0     0     1     0
     0     0    16     0
     0     0    15     0
     0     1     0     0
     0     0     1     0
     0     0    15     0
     1     1     0     1
     1     1     1     1
     1     1    16     1
     1     1    15     1
     0     1     0     0
     0     0     2     0
     1     3     0     0
     1     4     0     5

EMERGENCY!!! SECTOR 11 IS                    OVERLOADED

AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 11

THE SECTOR TO BE EXPANDED IS  6
```

Two sectors are returned to normal (sectors 6 and 13). Normally, NORPROS would execute more than once but its execution is limited for test purposes. Note that sectors 9-12 are in overload state one (normal coverage) but are shown as being overloaded. This is a result of the aircraft turn. These sectors were expanded previously and are now overloaded. As the sectors which cuased their expansion return to normal the new overloads will be dealt with.

```
   CURRENT OVERLOAD TABLE

     1      4      0      5
     1      2      0      0
     0      0      1      0
     0      0     16      0
     0      0     15      0
     0      0     11      0
     0      0      1      0
     0      0     15      0
     1      1      0      1
     1      1      1      1
     1      2     16      1
     1      1     15      1
     0      1      0      0
     0      0      2      0
     1      3      0      0
     1      4      0      5

EMERGENCY!!! SECTOR  9 IS                    OVERLOADED

AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS  9

THE SECTOR TO BE EXPANDED IS 13

   CURRENT OVERLOAD TABLE

     1      4      0      5
     1      2      0      0
     0      0      1      0
     0      0     16      0
     0      0     15      0
     0      0     11      0
     0      0      1      0
     0      0     13      0
     1      2      0      1
     1      1      1      1
     1      2     16      1
     1      1     15      1
     0      0      9      0
     0      0      2      0
     1      3      0      0
     1      4      0      5
```

EMERGENCY!!! SECTOR 12 IS OVERLOADED

CURRENT OVERLOAD TABLE

1	4	0	5
1	2	0	0
0	0	1	0
0	0	16	0
0	0	16	0
0	0	11	0
0	0	1	0
0	0	15	0
1	2	0	1
1	1	1	1
1	2	16	1
1	2	15	1
0	0	9	0
0	0	2	0
1	3	0	0
1	4	0	5

EMERGENCY!!! SECTOR 10 IS OVERLOADED

CURRENT OVERLOAD TABLE

1	4	0	5
1	2	0	0
0	0	1	0
0	0	16	0
0	0	16	0
0	0	11	0
0	0	1	0
0	0	15	0
1	2	0	1
1	2	1	1
1	2	16	1
1	2	15	1
0	0	9	0
0	0	2	0
1	3	0	0
1	4	0	5

NORPROS WILL NOW TRY TO REDUCE OVERLOADS

NORPROS WILL BE EXECUTED 2 TIMES

```
CURRENT OVERLOAD TABLE

  1     3     0     0
  1     2     0     2
  0     1     0     0
  0     1     0     0
  0     0    16     0
  0     0    11     0
  0     0     1     0
  0     0    15     0
  1     2     0     3
  1     2     1     3
  1     2    16     3
  1     2    15     3
  0     0     9     0
  0     0     2     0
  1     3     0     2
  1     3     0     0

EMERGENCY!!! SECTOR 10 IS                    OVERLOADED


AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK  IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 10

THE SECTOR TO BE EXPANDED IS  3
```

CURRENT OVERLOAD TABLE

1	3	0	0
1	2	0	2
0	0	10	0
0	1	0	0
0	0	16	0
0	0	11	0
0	0	1	0
0	0	15	0
1	2	0	3
1	3	1	3
1	2	16	3
1	2	15	3
0	0	9	0
0	0	2	0
1	3	0	2
1	3	0	0

EMERGENCY!!! SECTOR 11 IS OVERLOADED

AT THIS POINT, FSTRAM WOULD BE CALLED AND
WHEN CONTROL WAS PASSED BACK IT WOULD
RETURN CONTROL BACK TO NORPROS

THE OVERLOADED SECTOR IS 11

THE SECTOR TO BE EXPANDED IS 4

CURRENT OVERLOAD TABLE

1	3	0	0
1	2	0	2
0	0	10	0
0	0	11	0
0	0	16	0
0	0	11	0
0	0	1	0
0	0	15	0
1	2	0	3
1	3	1	3
1	3	16	3
1	2	15	3
0	0	9	0
0	0	2	0
1	3	0	2
1	3	0	0

The following test results are from a simulation of the logic array formatting subroutines and a special subroutine called DATAOUT which simulates the flow of data from an input table to a final destination. The overload table which is used reflects a normal state, i.e., the sixteen sectors share the system azimuth coverage equally.

INITIAL OVERLOAD TABLE

0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0

The first level logic array is formatted to steer data to the sectors equally. Each sector is sixteen azimuth increments (22.5 degrees) wide.

RAMTAB FOLLOWS

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
32	32	32	32	32	32	32	32
32	32	32	32	32	32	32	32
64	64	64	64	64	64	64	64
64	64	64	64	64	64	64	64
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
256	256	256	256	256	256	256	256
256	256	256	256	256	256	256	256
512	512	512	512	512	512	512	512
512	512	512	512	512	512	512	512
1024	1024	1024	1024	1024	1024	1024	1024
1024	1024	1024	1024	1024	1024	1024	1024
2048	2048	2048	2048	2048	2048	2048	2048
2048	2048	2048	2048	2048	2048	2048	2048
4096	4096	4096	4096	4096	4096	4096	4096
4096	4096	4096	4096	4096	4096	4096	4096
8192	8192	8192	8192	8192	8192	8192	8192
8192	8192	8192	8192	8192	8192	8192	8192
16384	16384	16384	16384	16384	16384	16384	16384
16384	16384	16384	16384	16384	16384	16384	16384
32768	32768	32768	32768	32768	32768	32768	32768
32768	32768	32768	32768	32768	32768	32768	32768

Normally, there would be one second level logic array for each sector. For test purposes, however, these separate tables are combined into one table which follows.

SRAMTAP FOLLOWS

1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2

4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8	1	1	1	1	2	2
2	2	4	4	4	4	8	8	8	8
1	1	1	1	2	2	2	2	4	4
4	4	8	8	8	8	1	1	1	1
2	2	2	2	4	4	4	4	8	8
8	8	1	1	1	1	2	2	2	2
4	4	4	4	8	8	8	8	1	1
1	1	2	2	2	2	4	4	4	4
8	8	8	8						

The test data to be steered via the logic arrays is contained in the table below.

INITIAL DATABUF TABLE

7	1000	2000	7
103	18000	4000	359
7	6000	6000	7
7	1000	8000	7
192	54000	10000	960
128	3000	12000	128
7	1000	14000	7
103	18000	16000	359
7	6000	18000	7
7	1000	20000	7
192	54000	22000	960
128	3000	24000	128
7	1000	26000	7
103	18000	28000	359
7	6000	30000	7
7	1000	32000	7

The following series of statements show how the pulse descriptor words were extracted from the input table and routed to their proper destinations.

```
                THE DATA IN DATABUF LINE   1  BELONGS TO SECTOR   1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   1  WAS CHOSEN
THE EMITTERS FREQUENCY IS    500  MHZ AND ITS TOA IS   2000  NS


                THE DATA IN DATABUF LINE   2  BELONGS TO SECTOR   7
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   7  WAS CHOSEN
THE EMITTERS FREQUENCY IS   9000  MHZ AND ITS TOA IS   4000  NS
```

```
                THE DATA IN DATABUF LINE    3  BELONGS TO SECTOR    1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   2  OF SECTOR    1  WAS CHOSEN
THE EMITTERS FREQUENCY IS    3000  MHZ AND ITS TOA IS    6000  NS


                THE DATA IN DATABUF LINE    4  BELONGS TO SECTOR    1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   2  OF SECTOR    1  WAS CHOSEN
THE EMITTERS FREQUENCY IS     500  MHZ AND ITS TOA IS    8000  NS


                THE DATA IN DATABUF LINE    5  BELONGS TO SECTOR   12
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   4  OF SECTOR   12  WAS CHOSEN
THE EMITTERS FREQUENCY IS   27000  MHZ AND ITS TOA IS   10000  NS


                THE DATA IN DATABUF LINE    6  BELONGS TO SECTOR    8
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   4  OF SECTOR    8  WAS CHOSEN
THE EMITTERS FREQUENCY IS    1500  MHZ AND ITS TOA IS   12000  NS


                THE DATA IN DATABUF LINE    7  BELONGS TO SECTOR    1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   2  OF SECTOR    1  WAS CHOSEN
THE EMITTERS FREQUENCY IS     500  MHZ AND ITS TOA IS   14000  NS


                THE DATA IN DATABUF LINE    8  BELONGS TO SECTOR    7
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   2  OF SECTOR    7  WAS CHOSEN
THE EMITTERS FREQUENCY IS    9000  MHZ AND ITS TOA IS   15000  NS


                THE DATA IN DATABUF LINE    9  BELONGS TO SECTOR    1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN   2  OF SECTOR    1  WAS CHOSEN
THE EMITTERS FREQUENCY IS    3000  MHZ AND ITS TOA IS   18000  NS
```

```
                THE DATA IN DATABUF LINE  10  BELONGS TO SECTOR   1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   1  WAS CHOSEN
THE EMITTERS FREQUENCY IS    500  MHZ AND ITS TOA IS  20000  NS


                THE DATA IN DATABUF LINE  11  BELONGS TO SECTOR  12
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  4  OF SECTOR  12  WAS CHOSEN
THE EMITTERS FREQUENCY IS  27000  MHZ AND ITS TOA IS  22000  NS


                THE DATA IN DATABUF LINE  12  BELONGS TO SECTOR   8
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  4  OF SECTOR   8  WAS CHOSEN
THE EMITTERS FREQUENCY IS   1500  MHZ AND ITS TOA IS  24000  NS


                THE DATA IN DATABUF LINE  13  BELONGS TO SECTOR   1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   1  WAS CHOSEN
THE EMITTERS FREQUENCY IS    500  MHZ AND ITS TOA IS  25000  NS


                THE DATA IN DATABUF LINE  14  BELONGS TO SECTOR   7
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   7  WAS CHOSEN
THE EMITTERS FREQUENCY IS   4000  MHZ AND ITS TOA IS  28000  NS


                THE DATA IN DATABUF LINE  15  BELONGS TO SECTOR   1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   1  WAS CHOSEN
THE EMITTERS FREQUENCY IS   3000  MHZ AND ITS TOA IS  30000  NS


                THE DATA IN DATABUF LINE  16  BELONGS TO SECTOR   1
                BIN SELECTION FOR THE CHOSEN SECTOR FOLLOWS
               BIN  2  OF SECTOR   1  WAS CHOSEN
THE EMITTERS FREQUENCY IS    500  MHZ AND ITS TOA IS  32000  NS
```

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/78-26	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A PARALLEL MICROPROCESSOR ARCHITECTURE FOR ELECTRONIC COUNTERMEASURES PROCESSING		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kenneth L. Henry Captain USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Strategic Systems System Program Office (ASD-YYE) Aeronautical Systems Division Wright-Patterson AFB, Ohio 45433		12. REPORT DATE December, 1978 13. NUMBER OF PAGES 157
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE:
IAW AFR 190-17.

JOSEPH P. HIPPS, Major, USAF
Director of Information 1-29-79

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Electronic Countermeasures Processing
Parallel Microprocessor Architecture
Data Steering
Parallelism

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The large number and wide variety of radio frequency emitters encountered in electronic warfare provide an enormous quantity of data for an electronic countermeasures system to process. Historically, all data has had to pass through a central processor for threat identification. Speed is the primary requirement for this ECM processor.

In this thesis the concept of parallelism was investigated as a method to increase the throughput of existing processors. Microprocessors were employed as the individual processing elements in a single data stream - multiple

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

instruction stream architecture. Data steering or vectoring was accomplished via the the use of programmed logic arrays stored in random access memories.

To test the feasibility of a parallel microprocessor architecture for ECM processing its basic building blocks were simulated. The necessary control logic, which is implemented in software in a master control processor, includes the capability to adapt the architecture to deal with the unpredictable nature of ECM threat data.

A fictitious scenario was developed to stimulate the data steering logic and test the master processor's capability to handle system overloads. Also, the actual ability of the system to process data was simulated through the use of fictitious radar parameters.

VITA

Kenneth Leslie Henry was born on 15 June 1951 in Linden, New Jersey. He graduated from Rahway High School in Rahway, New Jersey in 1969 and attended the United States Air Force Academy from which he received the degree of Bachelor of Science in Electrical Engineering in June 1973. Upon graduation, he received a commission as a Second Lieutenant in the United States Air Force. Immediately following commissioning, on 9 June 1973 he married the former Karen Elizabeth Billick in the chapel at Lemoore Naval Air Station, Lemoore, California. He served as a software and digital systems engineer with the B-1 System Program Office, Wright-Patterson Air Force Base, Ohio, from October 1973 until entering the School of Engineering, Air Force Institute of Technology, in January 1977.

Permanent address: 383 Russell Avenue
Rahway, New Jersey 07065